Foundations and Linear Methods

Machine Learning

Michael Wand

TA: Eric Alcaide

{michael.wand, eric.alcaide}@idsia.ch

Dalle Molle Institute for Artificial Intelligence Studies (IDSIA) USI - SUPSI

Fall Semester 2024

Contents

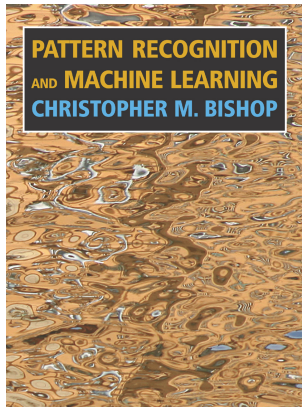
In this lecture we will cover the two basic tasks of supervised learning, as well as some fundamental aspects. Our schedule is as follows:

- Introduction to Regression
- Introduction to Classification
- Elements of the Foundations of Machine Learning

Besides introducing elementary examples of algorithms, we will also get to know *many basic concepts and definitions* which we will frequently encounter later on!

Attribution: Many figures taken from Bishop's Machine Learning textbook `https://www.microsoft.com/en-us/research/people/cmbishop/prml-book`.

Also highly recommended as a reference!

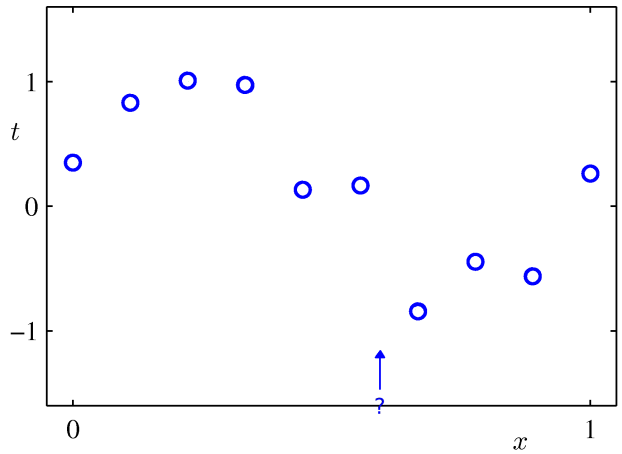


USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Introduction to Regression

Polynomial Fitting

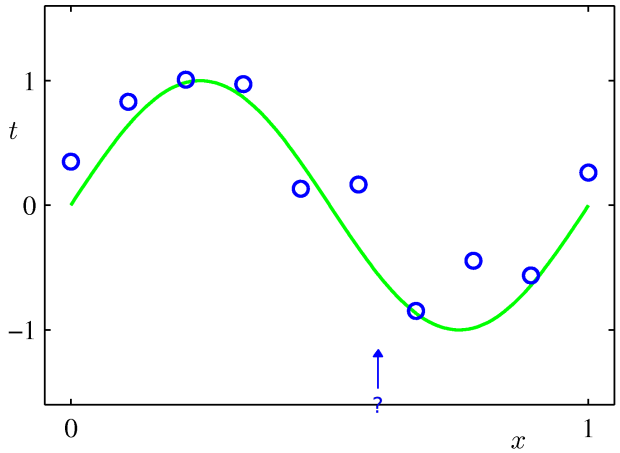
- We are given a training set of observations $\mathcal{D} = \{d_n\}_{n=1,\ldots,N} = \{(x_n, t_n)\}_{n=1,\ldots,N}$ (represented as blue circles).
- Task: Predict the value of the target variable t from the input variable x, for an *unknown* input x (blue '?').
- For now, both input and target are one-dimensional.
- Since we have observations which include the target, this is a supervised task.
- It is also a regression task (real-valued output).



Img src: Bishop, figure 1.2 (modified)

Polynomial Fitting

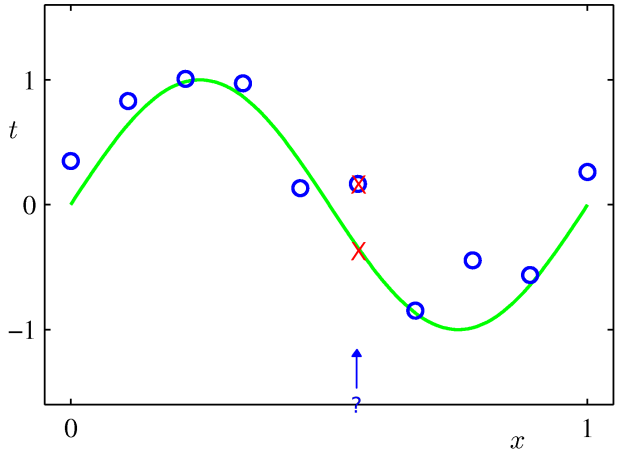
- Spoiler: the training data was generated from a sine wave with added (Gaussian) noise.
- ? Is there a "perfect" solution to our prediction problem?
- ? Can we find it, based on the available data? Can we approximate it?
- ? What do we expect from our solution?



Img src: Bishop, figure 1.2 (modified)

Polynomial Fitting

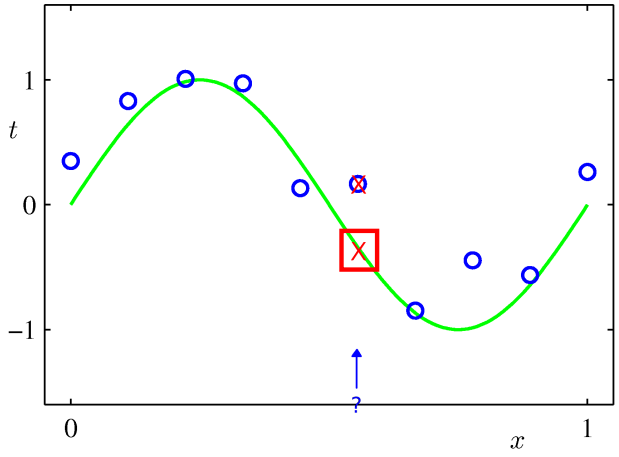
- Consider the example: The input x exactly matches one of the training samples.
- Which of the two 'X' is the "correct" prediction?



Img src: Bishop, figure 1.2 (modified)

Polynomial Fitting

- Consider the example: The input x exactly matches one of the training samples.
- Which of the two 'X' is the "correct" prediction?

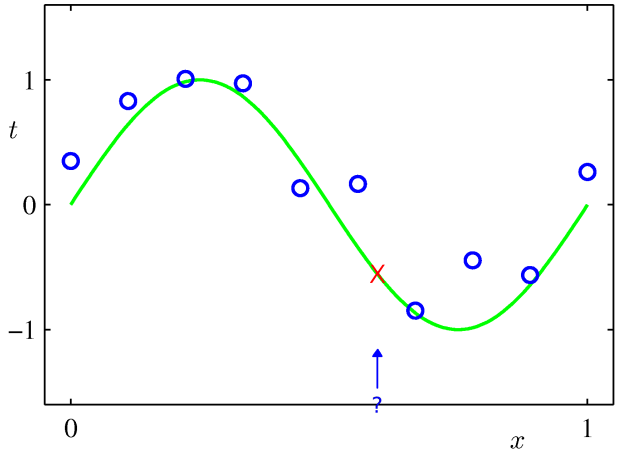


Img src: Bishop, figure 1.2 (modified)

- We usually aim at finding underlying structure in the task.
- In practical cases, the training data is frequently noisy.
- The lower 'X' is probably the better prediction.
- The answer may also depend on our knowledge or assumptions of the underlying data.

Polynomial Fitting

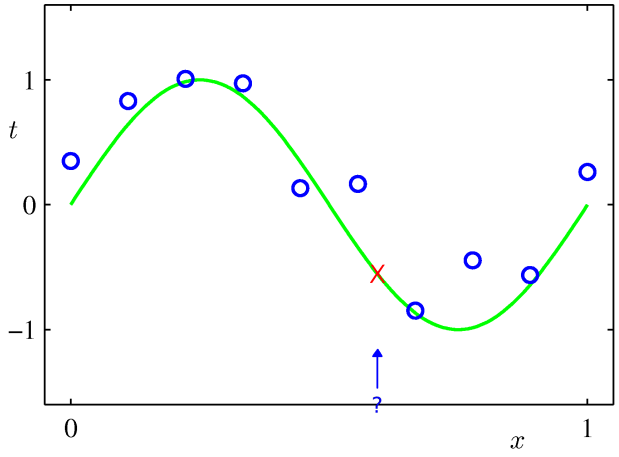
- What about the general case in which the input x does not match any training sample?
- Given our knowledge of the data, the prediction should be given by the red 'X'.



Img src: Bishop, figure 1.2 (modified)

Polynomial Fitting

- What about the general case in which the input x does not match any training sample?
- Given our knowledge of the data, the prediction should be given by the red 'X'.



Img src: Bishop, figure 1.2 (modified)

- This is called generalization: the system should be able to make reasonable predictions on *new* data, provided that this data is "similar" to the training data.
- Later on, we will formally define what "similarity" means.
- For this prediction we have assumed knowledge about the structure of the data.
- It turns out that we always have to make some kind of such assumption, otherwise *generalization is impossible* (No Free Lunch Theorem).

Polynomial Fitting

- Our goal is to approximate the underlying structure of the data.
- For this purpose we make a model assumption: we describe the relationship between input and target by a polynomial

$$y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j.$$

- After fitting, we wish to use y as estimator for t.
- We now need to fit the model to the input observations $\{(x_n, t_n)\}_{n=1,\ldots,N}$ by determining the coefficients $\mathbf{w} = \{w_j\}_{j=0,\ldots,M}$.
- (We also need to choose the order M, but for now assume that M is fixed.)
- Note that for the given task, this particular model will never exactly fit unseen data. (Why? Two reasons!)

Polynomial Fitting

- We define the Mean Squared Error (MSE) as

$$E(\mathbf{w}) = \frac{1}{N}\sum_n e_n(\mathbf{w}) = \frac{1}{N}\sum_n \frac{1}{2}(y(x_n, \mathbf{w}) - t_n)^2$$

  where $e_n(\mathbf{w}) = \frac{1}{2}(y(x_n, \mathbf{w}) - t_n)^2$ is the error for sample n.
- The MSE is our loss function and our training criterion: We aim to minimize it by choosing suitable parameters $\mathbf{w}$.
- We frequently do not write the explicit dependence of the error on $\mathbf{w}$.
- There are underlying reasons for choosing this error. For now, let us just say that it is computationally easy. We include the factor $\frac{1}{2}$ for later convenience.

Solving the Fitting Task

To summarize:

$$E = \frac{1}{N}\sum_n e_n, \qquad e_n = \frac{1}{2}(y(x_n, \mathbf{w}) - t_n)^2, \qquad y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j = \mathbf{w}^T \boldsymbol{\phi}(x).$$

with $\boldsymbol{\phi}(x) = (\phi_0(x), \ldots, \phi_M(x))^T = (x^0, x^1, \ldots, x^M)^T$ (useful later).

We compute the derivative of the error:

$$\frac{dE}{d\mathbf{w}} = \left(\frac{\partial E}{\partial w_0}, \ldots, \frac{\partial E}{\partial w_M}\right)$$

with

$$\frac{\partial E}{\partial w_j} = \frac{1}{N}\sum_n \frac{\partial e_n}{\partial w_j} = \frac{1}{N}\sum_n (y(x_n, \mathbf{w}) - t_n)\cdot\frac{\partial y(x_n, \mathbf{w})}{\partial w_j} = \frac{1}{N}\sum_n (\mathbf{w}^T\boldsymbol{\phi}(x_n) - t_n)\cdot\phi_j(x_n)$$

and use matrix calculus to simplify:

$$\frac{dE}{d\mathbf{w}} = \frac{1}{N}\left(\mathbf{w}^T\sum_n \boldsymbol{\phi}(x_n)\boldsymbol{\phi}(x_n)^T - \sum_n t_n\boldsymbol{\phi}(x_n)^T\right).$$

Solving the Fitting Task

- Using the *design matrix*

$$\mathbf{\Phi} = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_M(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_M(x_N) \end{pmatrix}$$

  and the target vector $\mathcal{T}^T = (t_1, \ldots, t_N)^T$, we can further simplify:

$$\frac{dE}{d\mathbf{w}} = \frac{1}{N}\left(\mathbf{w}^T(\Phi^T\Phi) - \mathcal{T}^T\Phi\right).$$

- The fitting task is solved by minimizing the error, which we do by setting the derivative of the error to zero:

$$0 \stackrel{!}{=} \frac{dE}{d\mathbf{w}} = \frac{1}{N}\left(\mathbf{w}^T(\Phi^T\Phi) - \mathcal{T}^T\Phi\right).$$

Solving the Fitting Task

- Cancelling the factor $\frac{1}{N}$ and transposing, the equation

$$0 \stackrel{!}{=} \mathbf{w}^T \Phi^T \Phi - \mathcal{T}^T \Phi \Leftrightarrow 0 \stackrel{!}{=} \Phi^T \Phi \mathbf{w} - \Phi^T \mathcal{T}$$

  is a linear system of $(M+1)$ equations for $(M+1)$ variables, so we will assume that there is a unique solution. (Q: What is the condition for a solution to exist?)
- It can be seen easily that the solution is given by

$$\mathbf{w}_{\text{min}} = (\Phi^T \Phi)^{-1} \Phi^T \mathcal{T}.$$

- The matrix $(\Phi^T \Phi)^{-1} \Phi^T$ is called the Moore-Penrose Pseudo-Inverse of the matrix Φ (Φ is in general is not a square matrix).
- If Φ is square and invertible, the Moore-Penrose Pseudo-Inverse coincides with Φ^{-1} (the proper inverse).

Polynomial Fitting

- We have solved our first regression task!
- Now we can use $y(x, \mathbf{w}_{\text{min}})$ to predict targets t from inputs x.
- Key observation: we had to solve an equation which was nonlinear in the inputs, but *linear* in the parameters, i.e. y depends linearly on $\mathbf{w}$.

$\Rightarrow$ This allowed us to find a closed-form solution!

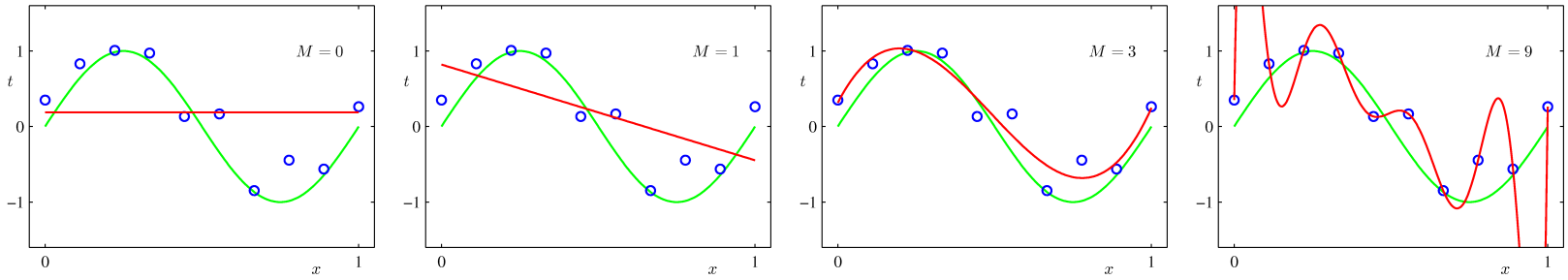
- Unfortunately, finding solutions for *nonlinear* models is not so easy. During this lecture we will get to know several ways to approximate such solutions!
- Also remember that this is the "best" solution in terms of the criterion which we used, but it is not necessarily a perfect solution, and there may be other solutions for other criteria!

Polynomial Fitting: Order

So how should we choose the order M of the fitting polynomial?
Remember our goal: predict the target variable for *new* data points.
Here are some fitted curves (in red) for different polynomial orders M.

Polynomial Fitting: Order

So how should we choose the order M of the fitting polynomial?
Remember our goal: predict the target variable for *new* data points.
Here are some fitted curves (in red) for different polynomial orders M.

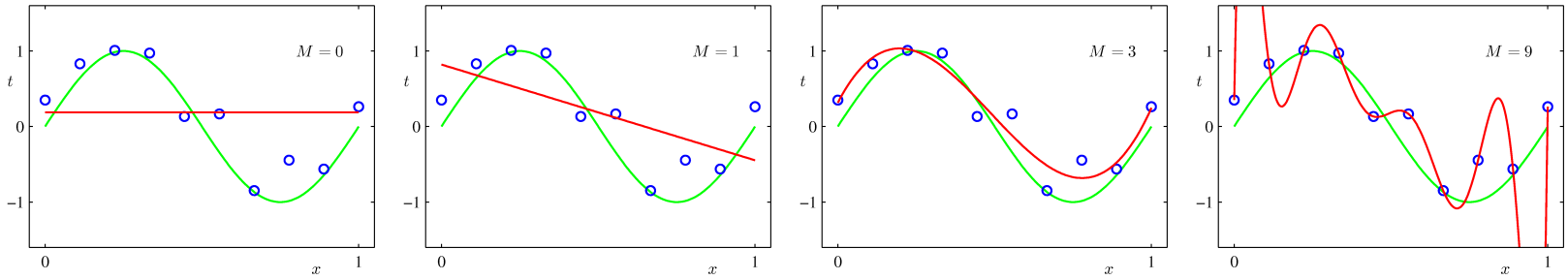


Img src: Bishop, figure 1.4

Which one would you choose?

Polynomial Fitting: Order

So how should we choose the order M of the fitting polynomial?
Remember our goal: predict the target variable for *new* data points.
Here are some fitted curves (in red) for different polynomial orders M.

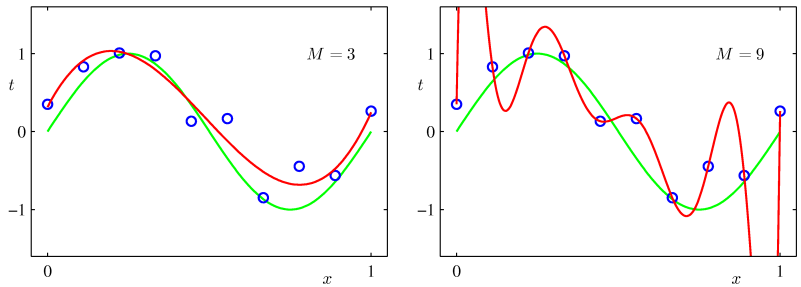


Img src: Bishop, figure 1.4

Which one would you choose?

- Clearly, the linear approximations with orders 0 and 1 do not well represent the data.
- These orders are too low for the task: these polynomials are not expressive enough, they underfit the data.

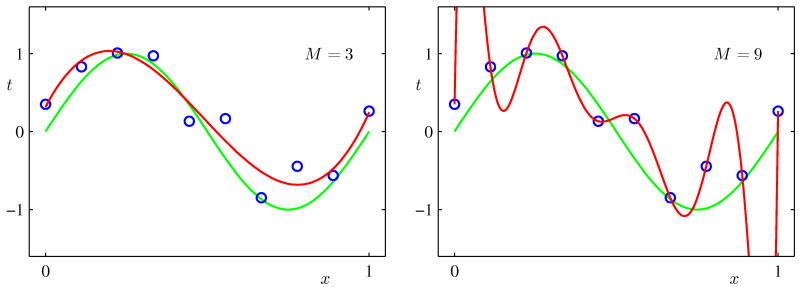
Polynomial Fitting: Order



Img src: Bishop, figure 1.4

- The polynomial of order 9 exactly fits the training samples (why?), but does not recover the underlying structure.
- It overfits the training data, yielding bad predictions for new data points.
- One can also say that it has been tuned to the random noise present in the data.

Polynomial Fitting: Order



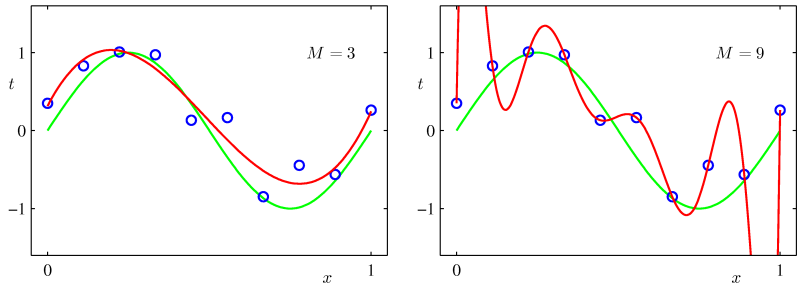
Img src: Bishop, figure 1.4

- The polynomial of order 9 exactly fits the training samples (why?), but does not recover the underlying structure.
- It overfits the training data, yielding bad predictions for new data points.
- One can also say that it has been tuned to the random noise present in the data.
- The overfitted polynomial has very large coefficients.

Table: Coefficients of the fitted polynomial

Order	M=1	M=3	M=9
w_0	0.82	0.31	0.35
w_1	-1.27	7.99	232.37
w_2		-25.43	-5321.83
w_3		17.37	48568.31
w_4			-231639.30
w_5			640042.26
w_6			-1061800.52
w_7			1042400.18
w_8			-557682.99
w_9			125201.43

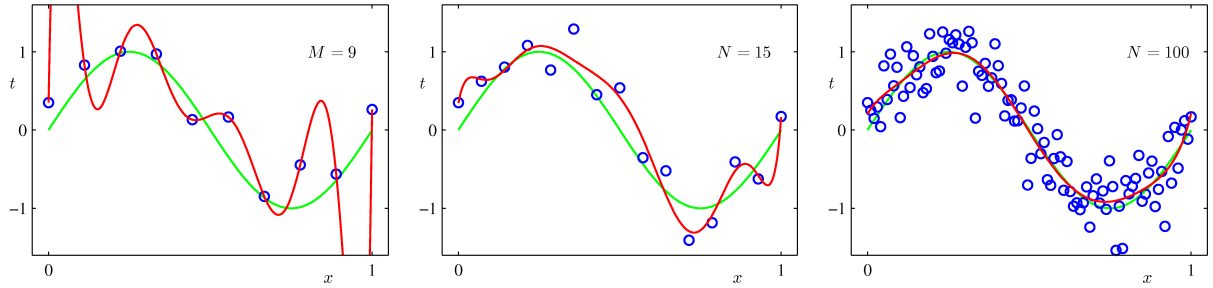
Polynomial Fitting: Order



Img src: Bishop, figure 1.4

- Within the set of models which we considered, the order 3 polynomial is optimal.
  - ➔ Even though the training error is higher than for higher-order polynomials.
  - ➔ A perfect solution with a polynomial is not possible (since the "true" shape of the data is not a polynomial, and since the data has noise).

Reducing Overfitting



Img src: Bishop, figure 1.4 and 1.6

- Easiest way to reduce overfitting: Get more training data
- Figures: estimated regression polynomial for $M = 9$ with $N = 10, 15, 100$ data points
- In the latter case, the curve does not overfit: better generalization
- Don't have enough data? Maybe you can artificially create it
  - ➔ by injecting random noise to samples
  - ➔ by domain-specific transformations (e.g. in image recognition, you could slightly deform images without changing their content)
  - ➔ relevant for all machine learning tasks (classification, regression, and others), and for a variety of methods

Reducing Overfitting

- Limiting the number of model parameters (in this case, polynomial coefficients) helps against overfitting.
- Parameters can also be constrained by penalizing their absolute value.
- In the case of linear regression, optimize a modified error function with a suitable regularization term, e.g.:

$$\tilde{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}(y(x_n,\mathbf{w})-t_n)^2+\frac{\lambda}{2}||\mathbf{w}||_2^2 = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{2}(y(x_n,\mathbf{w})-t_n)^2+\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- λ determines the relative weight of the regularization term.
- This regularization is called L_2 regularization.
- The solution has a closed form (Ridge Regression):

$$\mathbf{w}_{\text{min}} = (\lambda\mathbf{I}+\mathbf{\Phi}^\mathbf{T}\mathbf{\Phi})^{-1}\mathbf{\Phi}^\mathbf{T}\mathcal{T}.$$

- Other forms of regularization exist and are regularly used.

Features

- Important observation: The solution depends only on the features, i.e. the elements of the design matrix:

$$\mathbf{\Phi} = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_M(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_M(x_N) \end{pmatrix}$$

- We had $\phi_j(x) = x^j$, but we see that can choose features $\phi_j(x)$ *arbitrarily*, leading to fitting with arbitrary functions using the same method as before.
- The choice of features often depends on knowledge of the task.
- In general, computing features amounts to preprocessing the data in a way to make relevant information accessible, and to remove irrelevant content.

Multivariate Regression / Curse of Dimensionality

- Assume we have multiple input variables $x^{(1)}, x^{(2)}, \ldots, x^{(L)}$.
- For polynomial fitting, we have to take into account all possible products between variables, up to the desired order M:

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{\ell} w_1^{(\ell)} x^{(\ell)} + \sum_{\substack{\ell_1, \ell_2 \\ \ell_1 \leq \ell 2}} w_2^{(\ell_1, \ell_2)} x^{(\ell_1)} x^{(\ell_2)} +$$

$$+ \ldots + \sum_{\substack{\ell_1, \ell_2, \ldots, \ell_M \\ \ell_1 \leq \ell_2 \leq \ldots \leq \ell_M}} w_M^{(\ell_1, \ell_2, \ldots, \ell_M)} x^{(\ell_1)} x^{(\ell_2)} \ldots x^{(\ell_M)}.$$

- Thus we need to estimate $O(M^L)$ coefficients, which may be inefficient and can cause severe overfitting.

$\Rightarrow$ This problem in high-dimensional input spaces is called the Curse of Dimensionality: the number of model parameters rises exponentially with the input dimension!

Multivariate Regression / Curse of Dimensionality

Fortunately, dimensionality problems can be tackled:

- Real data often lies in a subspace of lower dimensionality
- Example: An image of $100 \cdot 100$ pixels has 10000 dimensions. . . but how many theoretically possible images show anything sensible?

$\Rightarrow$ Use *dimensionality reduction* techniques on the data.

$\Rightarrow$ Define a suitable set of features: $\phi_k(x_1, x_2, \ldots, x_L)$, where the number of features can be much smaller than L^M.

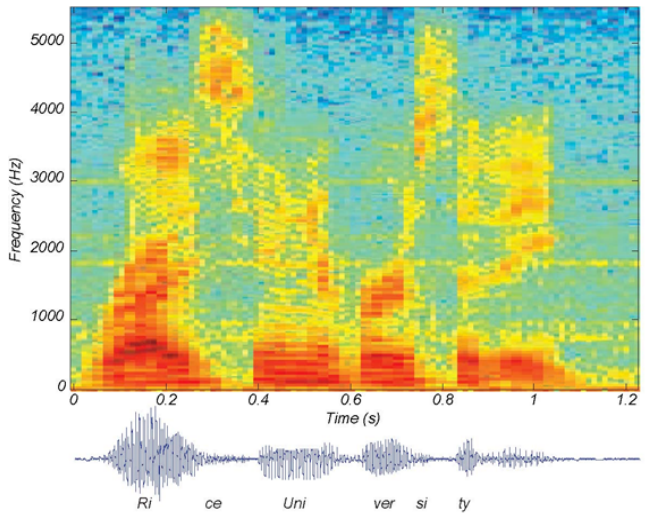
$\Rightarrow$ In realistic tasks, features can be very complex, and feature optimization plays a crucial role.

- Neural networks can intrinsically deals with high-dimensional input and often do not require sophisticated features.

Multidimensional output: less of a problem, can estimate output variables independently (but versatile algorithms estimate them jointly).

Examples of Features

- Features play a central role in almost all machine learning applications:
  - ➔ they reduce the dimensionality of the data
  - ➔ they make hidden information accessible
  - ➔ they suppress irrelevant information.
- Example: The speech *spectrogram*, computed by taking the Fourier transformation of short speech units (frames)



Img src: https://www.opentextbooks.org.hk/ditatopic/9759

The Linear Regression Model

- Step by step, we have gotten to know a technique which is called Linear Regression.
- Idea: model the relationship between input features and targets as a linear equation:

$$\mathcal{T} = \mathbf{W}\phi + \epsilon,$$

  where the vector ϵ is the error term (also called noise).
- We aim to minimize the noise. If we use the MSE loss to measure the error, we have already derived the closed-form solution for this task, including a regularization term if desired.
- However, other strategies for solving linear regression are possible.
- The features are predefined and could be generic (like polynomial coefficients), or application-specific. Note that the features do *not* need to depend linearly on the raw input data.

Regression: Summary

You have learned about

- fitting a regression model which is linear in its parameters: an exact solution
- examples of overfitting and underfitting, regularization
- the role of features
- issues of dimensionality.

Regression: Summary

We summarize the following important definitions:

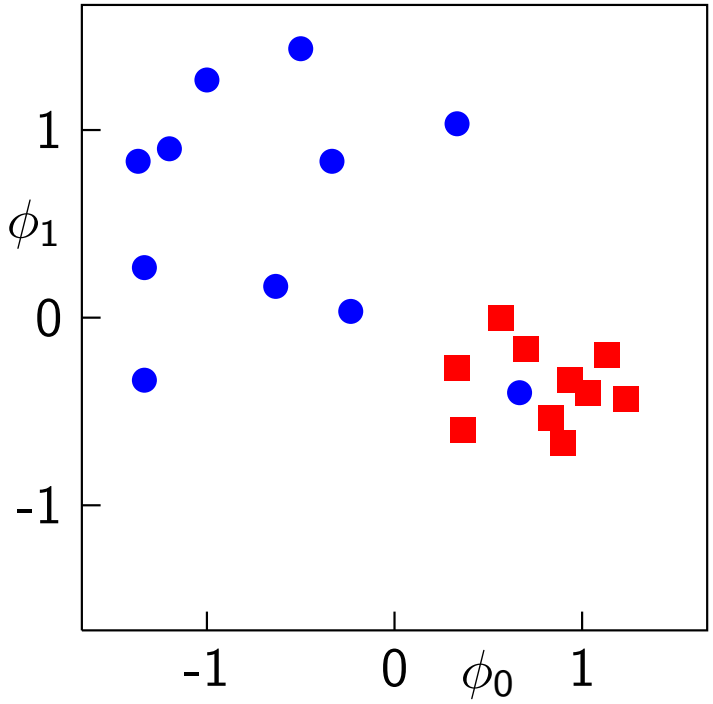
- generalization: the capability of a system to adapt to novel, unseen data
- underfitting: when the model does not reflect the structure of the training data
- overfitting: when the model has learned the structure of the training data very well, but fails at generalizing to novel test data
- regularization: changing the training criterion of a system to improve generalization.

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Introduction to Classification

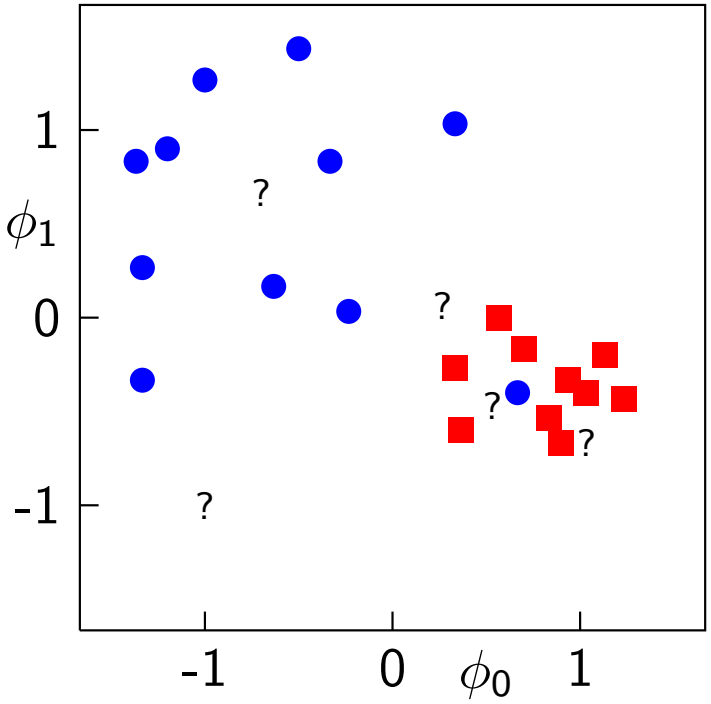
Classification

- Classification: Assign categorical values to input data points.
- Here we have two classes, e.g. $0 =$ ■ , $1 =$ ●.
- Multiple classes: often use 1-of-K (or one-hot) coding:
  - ➔ $(1, 0, 0, ...)$ for class 0,
  - ➔ $(0, 1, 0, ...)$ for class 1, etc.
- As before, we assume we have some training data and some test data.
- Have a look at the training data given for a two-class problem with two features ϕ_0 and ϕ_1.



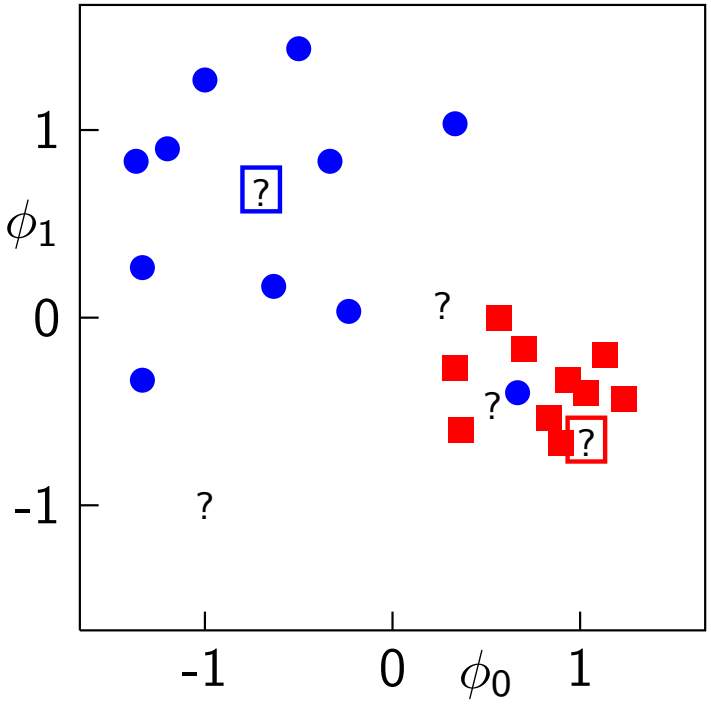
Classification

- How would you assign classes to the '?' test data points?
- If in some cases you find the answer "obvious", think a second why.



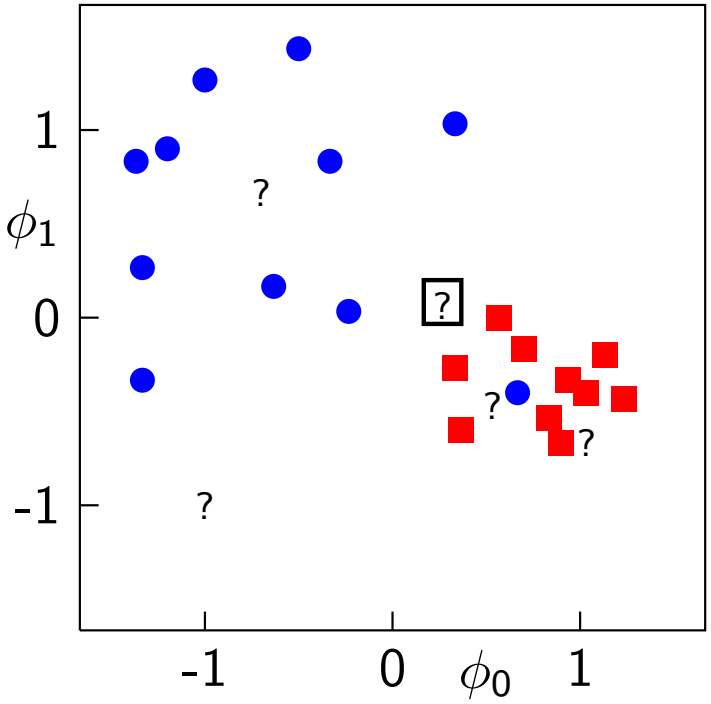
Classification

- How would you assign classes to the '?' test data points?
- Two easy cases: follow class assignments of surrounding data point(s)



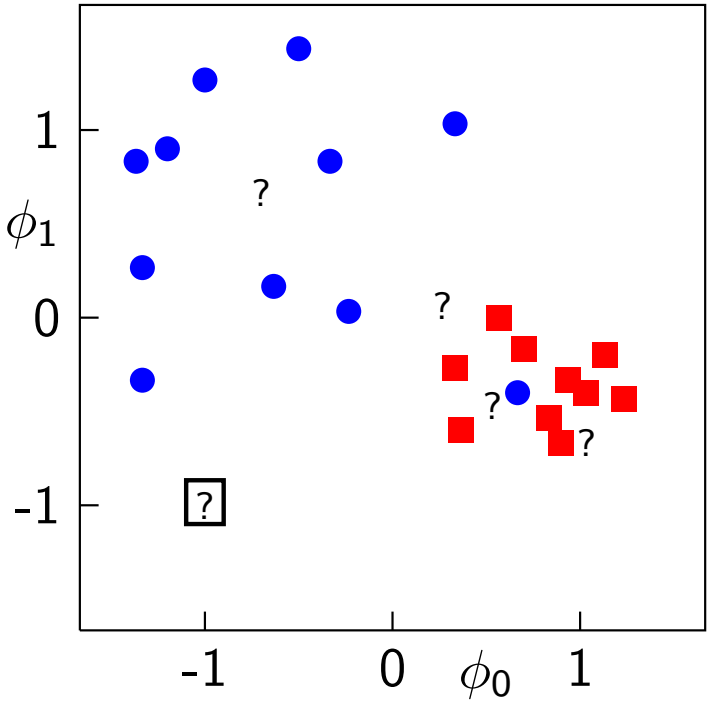
Classification

- How would you assign classes to the '?' test data points?
- This one is closer to the ■ class, but note that the ■ are much more compact than the ●. Could be ambiguous.



Classification

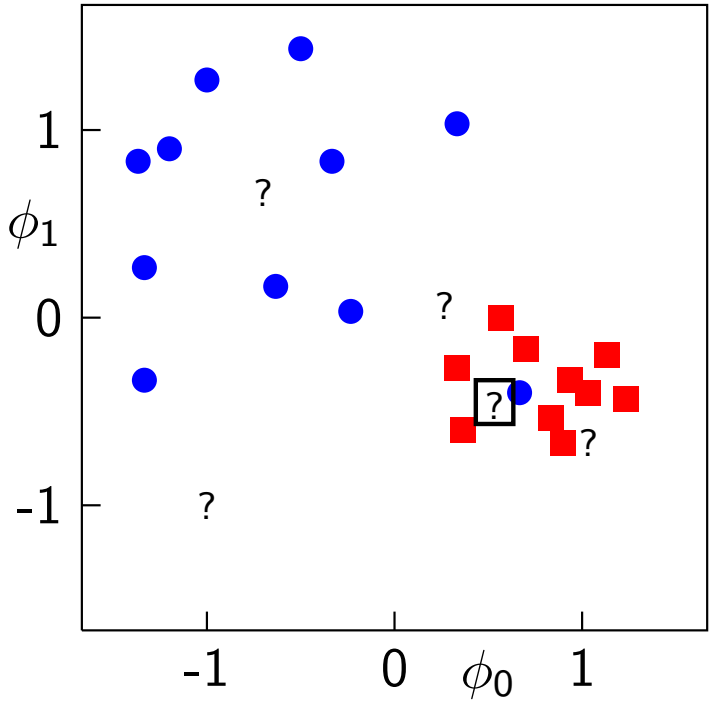
- How would you assign classes to the '?' test data points?
- This one is far away from any of the classes. Probably should be a ●, but realistically, the training data does not prepare us well for classifying this particular sample.



Classification

- How would you assign classes to the '?' test data points?
- This one is close to a ●, which seems however an outlier - a nontypical example, maybe due to data noise.
- Remember that data is never perfect, and that ambiguities are quite normal!
- Even if it is not an outlier, we can intuitively say that the "probability" of ■ seems higher than ●.
- Probabilistic modeling will allow us to formalize this idea.

$\Rightarrow$ In this (simple) case, we can solve the problem by considering multiple neighbors (not just the closest one).



K-Nearest-Neighbors classifier

Define the simple, nonparametric k-Nearest-Neighbors (kNN) classifier:

- for a test sample x, consider the k nearest training samples (for fixed k)
- determine the class assignment for x by "majority vote" (i.e. it corresponds to the class of the majority of the k nearest training samples)
- one could also weigh the influence of the k nearest neighbors (e.g. by distance)

The algorithm is called nonparametric because works directly with the data, as opposed to parametric methods like linear regression which are based on learning parameters of a model.
What do you think are the problems of this approach?

K-Nearest-Neighbors classifier - Problems

Here are some major problems of the kNN classifier:

- requires to store all training data, and to compute distances between test sample and all training samples
- does not discover structure (e.g. shape, "compactness" of the classes)
- very sensitive to outliers
- problematic in high-dimensional feature spaces (Curse of Dimensionality: samples do not cover the space well)
- difficult to define suitable metric of closeness.

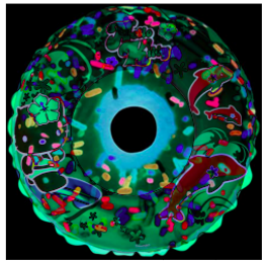
K-Nearest-Neighbors classifier - Problems

- Consider a k-nearest-neighbors classifier with which we want to distinguish a swim tyre and a donut.
- Images are represented at 250x250 pixels, 3 colors, ranging from 0..255.
- Thus, 250x250x3 = 187500 features in raw pixel space.

Img Source: Found on Amazon (yes, also the donut)

K-Nearest-Neighbors classifier - Problems

- Absolute difference (=distance) between swim tyre and donut, in raw pixel space: average ≈ 45.



- Absolute difference between rescaled donut and *shifted identical donut*, in raw pixel space: average ≈ 146.

a) Input image 1 b) Input image 2 c) absolute difference

- This cannot be a good way to perform (image) classification.

K-Nearest-Neighbors classifier - Problems

- In practice, one would define suitable features (search internet for HOG, SIFT, etc. if you are interested), and one would normalize (e.g. center) the images.
- But what about rotation, scaling, photos from different angles . . . ?
- Clearly, complex realistic cases cannot be solved this way. Need a *much* more versatile approach.
- We will get to know such approaches in this lecture. They are always based on some kind of model, i.e. they are *parametric*.

The Linear Classifier

- Parametric classifier: structure of data gets summarized by a set of parameters of (usually) fixed size, independent of the number of training samples
- Two fundamental concepts:
  - ➔ Discriminant function: A function (with trainable parameters) which assigns a class to a test sample

$$\mathcal{C} = f(\phi)$$

  - ➔ Probabilistic modeling: A function models the probabilities of a test sample belonging to any class

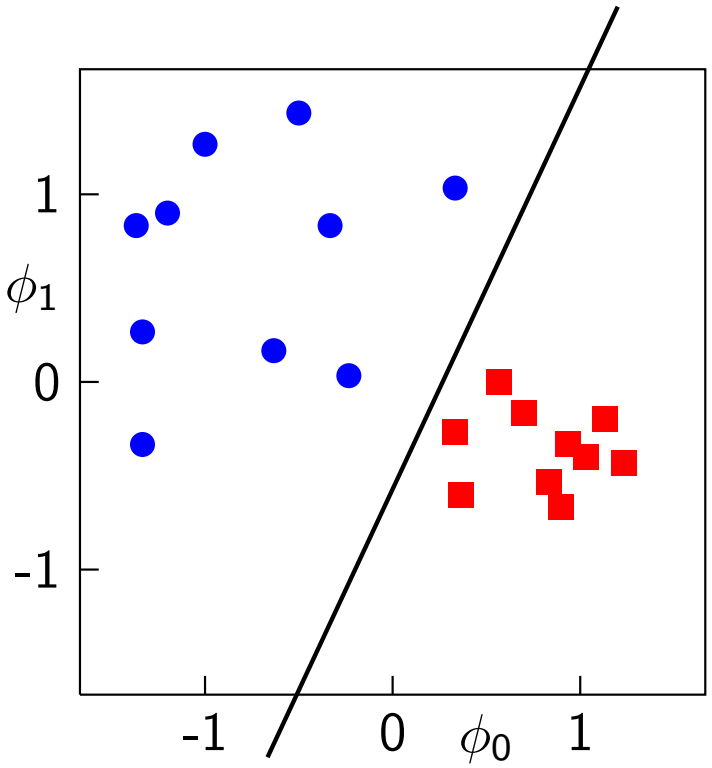
$$p(\mathcal{C}_k|\phi) = f(\phi, \mathcal{C}_k)$$

- It is easy to compute the discriminant function from a probabilistic model (take the class whose probability is highest).

The Linear Classifier

- Consider the example below (the outlier has been removed).
- Most simple discriminant function: a hyperplane in feature space.

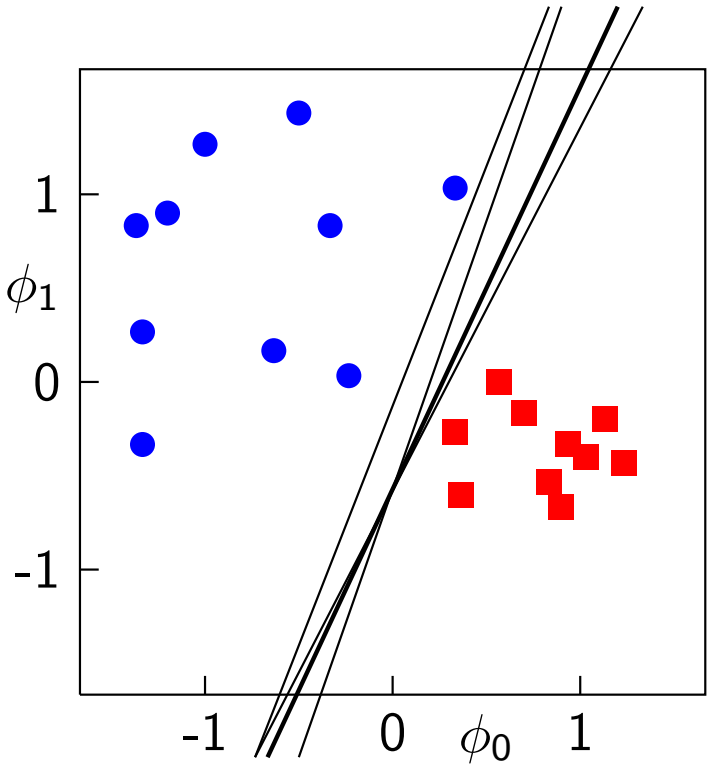
$\Rightarrow$ In the two-dimensional case: a straight line.



The Linear Classifier

- Consider the example below (the outlier has been removed).
- Most simple discriminant function: a hyperplane in feature space.

$\Rightarrow$ In the two-dimensional case: a straight line.

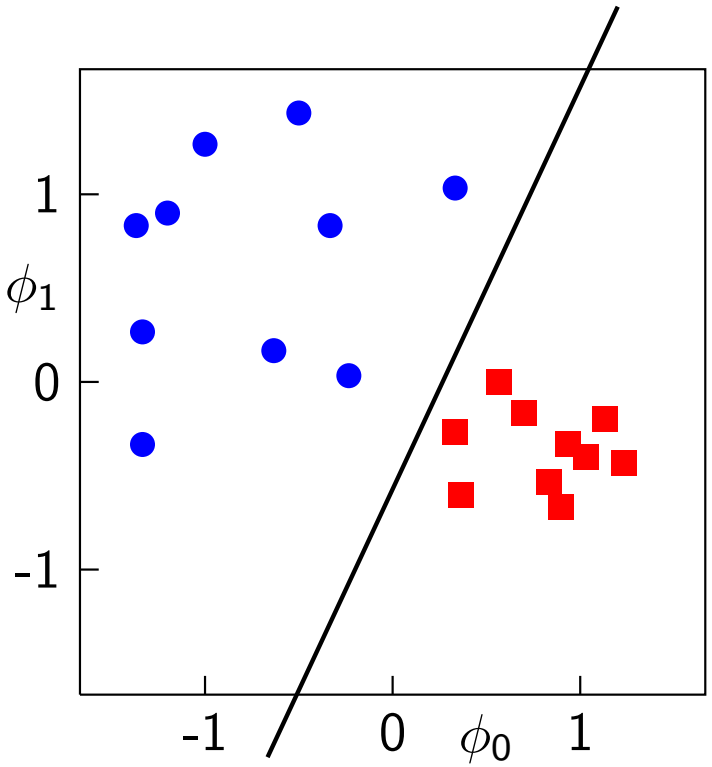


- In the given example, many solutions are possible.
  - ➔ We will see later how to choose a good one.
  - ➔ We will see later what to do if there is no perfect solution.

The Linear Classifier

The discriminant function takes the form

$$y(\phi) = \mathbf{w}^T\phi + w_0 \qquad \text{with} \qquad \mathcal{C}(\phi) = \begin{cases} \mathcal{C}_0, & \text{if } y(\phi) \geq 0 \\ \mathcal{C}_1, & \text{if } y(\phi) < 0 \end{cases}$$

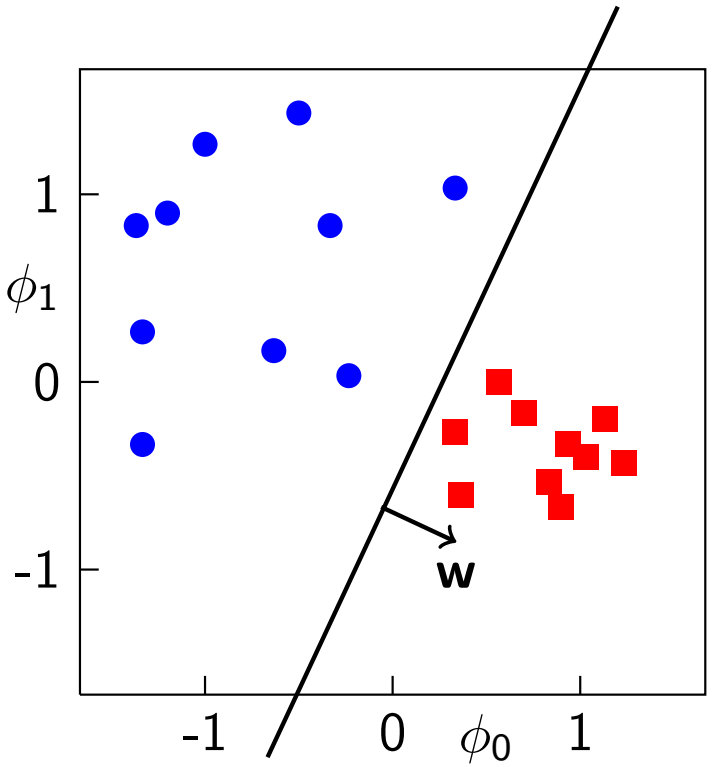


$\mathbf{w}$ is called a weight vector, w_0 is the bias. The hyperplane with the equation $\mathbf{w}^T\phi + w_0 = 0$ is called decision boundary, it separates the space into decision regions.

The Linear Classifier

The discriminant function takes the form

$$y(\phi) = \mathbf{w}^T \phi + w_0 \qquad \text{with} \qquad \mathcal{C}(\phi) = \begin{cases} \mathcal{C}_0, & \text{if } y(\phi) \geq 0 \\ \mathcal{C}_1, & \text{if } y(\phi) < 0 \end{cases}$$



Note that $\mathbf{w}$ is orthogonal to the decision boundary. The distance of any point x to the decision boundary is $\frac{y(\mathbf{x})}{||\mathbf{w}||}$, in particular, the distance of the decision boundary to the origin (0,0) is $\frac{|w_0|}{||\mathbf{w}||}$.

The Linear Classifier - Observations

- We now have a mathematical formulation for the classification problem: the decision regions are separated by a hyperplane parametrized by $\mathbf{w}$ and w_0.
  - ➔ We will get to know different ways to choose $\mathbf{w}$ and w_0.
- The decision regions of the kNN classifier have a much more complex shape!
- The parametrized model is tendentially robust against outliers.

The Linear Classifier - Observations

- We now have a mathematical formulation for the classification problem: the decision regions are separated by a hyperplane parametrized by $\mathbf{w}$ and w_0.
  - ➔ We will get to know different ways to choose $\mathbf{w}$ and w_0.
- The decision regions of the kNN classifier have a much more complex shape!
- The parametrized model is tendentially robust against outliers.
- This comes at the price of lower flexibility: This classifier is prone to drastic *underfitting*.
- We could get a better fit by choosing a decision boundary described by a polynomial

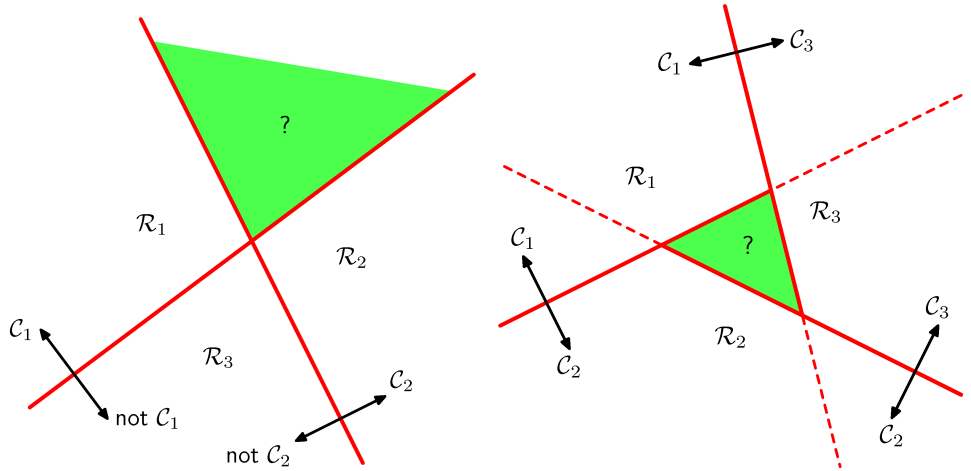
$\Rightarrow$ just like for regression, we can instead define suitable nonlinear features, so that the classifier remains linear in its parameters.

Multiple Classes

How to extend linear classification to $K > 2$ classes?

- *One-vs-rest* classification (left figure): use K classifiers, each of which distinguishes points in class k from points not in class k.
- *One-vs-one* classification (right figure): use $K(K-1)/2$ classifiers, one for each pair of classes.

Each of these methods leads to ambiguous areas (shaded green).



Img src: Bishop, figure 4.2

Multiple Classes

How to extend linear classification to $K > 2$ classes?

- Better solution: Use K linear functions of the form

$$y_k(\phi) = \mathbf{w}_k^T \phi + w_{k0}$$

and assign a data point $\phi(x)$ to class $\mathcal{C}_k$ if

$$y_k(\phi(x)) > y_j(\phi(x))$$

for all $j \neq k$.

- This method avoids ambiguous regions, all decision regions are singly connected and convex.

Logistic Regression

We present one specific way to define a linear classifier.

- We would like to use the ideas we have developed for linear regression to perform classification.
- Assume that our training data consists of observations $\{(\phi_n, t_n)\}_{n=1,\ldots,N}$, where t_n only takes the values 0 and 1. Train a regressor on this data.
- While performing "regression for classification" in this way is technically possible, it leads to issues:
  - ➔ What does it mean if the estimate of t for an input data point is not equal 0 or 1?
  - ➔ Assume a data point belonging to class 1. The MSE loss will be the same regardless of whether the estimate of t is 0.6 or 1.4 – do you think this makes sense?
- Clearly, we need to deal with the regression output in a different way.

Logistic Regression

- We bridge the gap from regression to classification by imposing a simple probabilistic model.
- Idea: We assume that each data point has a certain *probability* of belonging to class 0 or 1:

$$p(\mathcal{C}_1|\phi) = y(\phi)$$
$$p(\mathcal{C}_0|\phi) = 1 - y(\phi)$$

- From the probabilistic model, we derive the discriminant function:

$$\mathcal{C}(\phi) = 1 \qquad \text{if } p(\mathcal{C}_1|\phi) = y(\phi) \geq 0.5$$
$$\mathcal{C}(\phi) = 0 \qquad \text{if } p(\mathcal{C}_1|\phi) = y(\phi) < 0.5.$$

- With this model we can also express degrees of uncertainty about classification results!

Logistic Regression

- We derive the function y from Linear Regression.
- We make the output of regression probabilistic by passing it through a "squeezing function" which normalizes the real-valued output to the range $[0.0, 1.0]$.
- The classical choice of the squeezing function is the logistic function or logistic sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

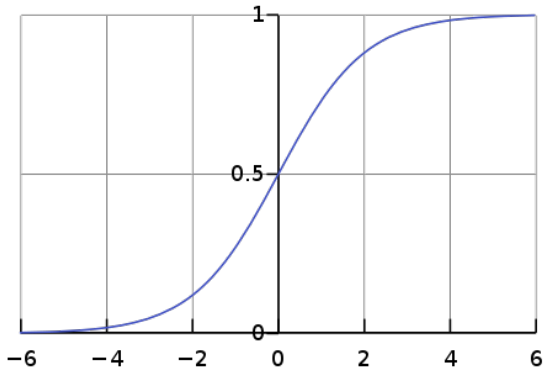


Image source: Wikipedia, *Sigmoid Function*

Logistic Regression

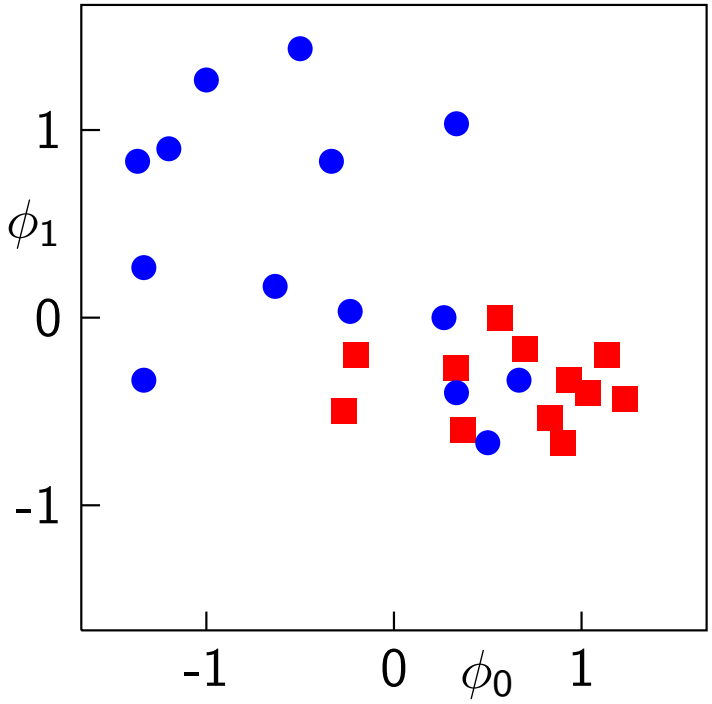
- We define

$$\tilde{y}(\phi) = \mathbf{w}^T \phi + w_0$$
$$y(\phi) = \sigma(\tilde{y}(\phi)).$$

- Taking $y = y(\phi) = y(\phi, \mathbf{w}, w_0)$ as the probability that a sample ϕ belongs to class 1, we see that
  - ➔ greater absolute values of $y(\phi)$ imply less uncertainty about the classification of the sample ϕ,
  - ➔ the decision boundary between the classes is given by the hyperplane $\tilde{y}(\phi) = 0$.
- This model is called Logistic Regression.

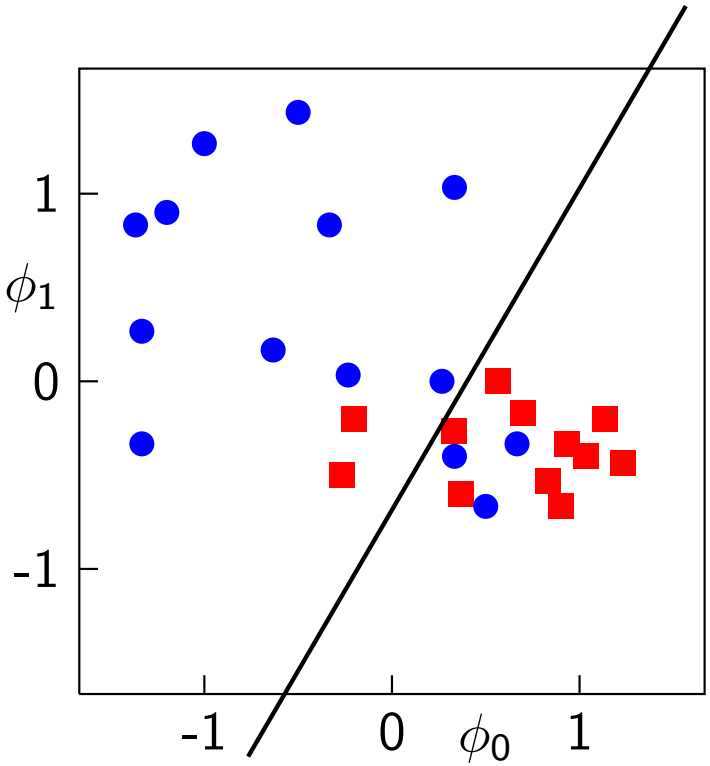
Logistic Regression

- Consider the following example task. Note that the classes overlap.



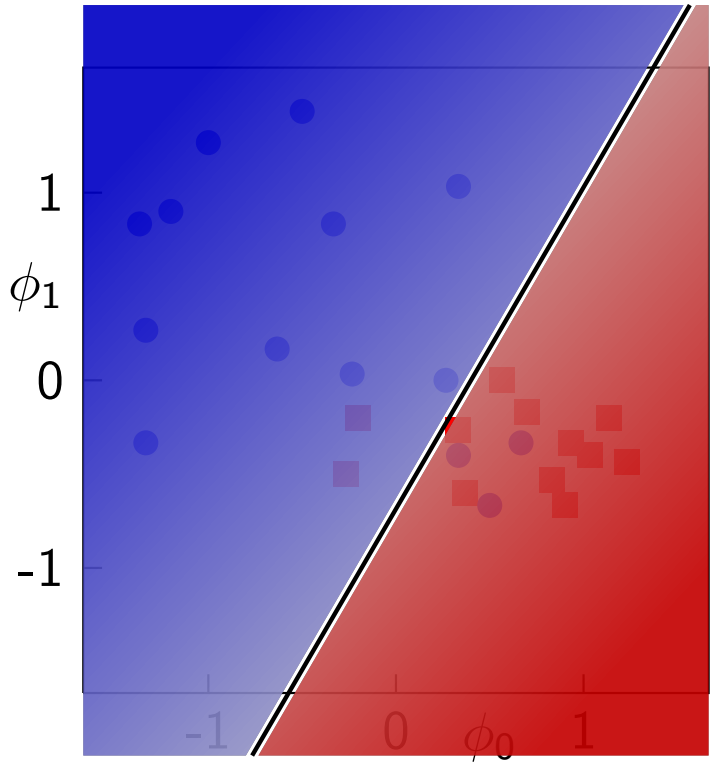
Logistic Regression

- Consider the following example task. Note that the classes overlap.
- This is the decision boundary which the model finds.



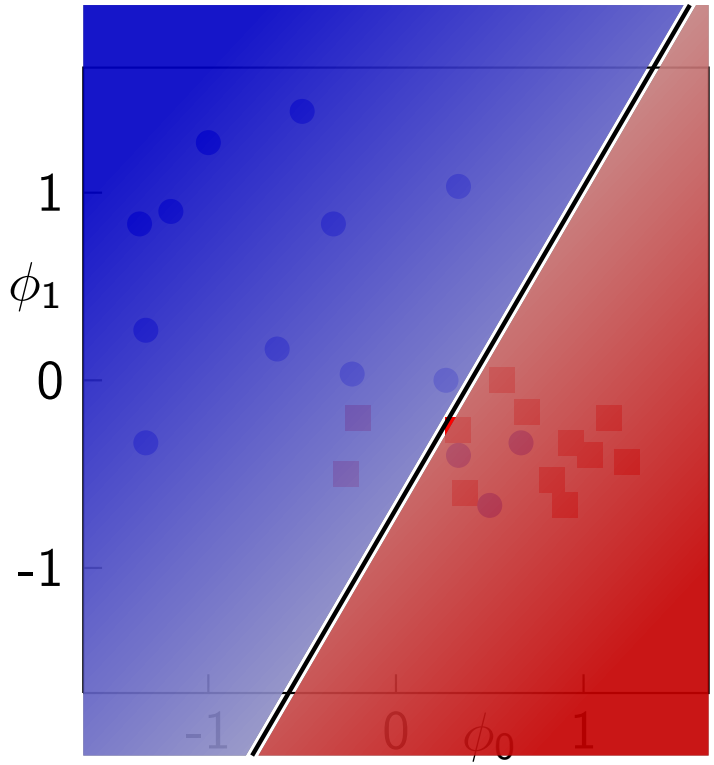
Logistic Regression

- Consider the following example task. Note that the classes overlap.
- This is the decision boundary which the model finds.
- Items which are more distant from the decision boundary are assigned to their respective classes with higher probability.



Logistic Regression

- Consider the following example task. Note that the classes overlap.
- This is the decision boundary which the model finds.
- Items which are more distant from the decision boundary are assigned to their respective classes with higher probability.
- Clearly, the model does not classify the data perfectly.



Logistic Regression

- We need a training criterion for logistic regression.
- This criterion will be based on probabilities, just like the model itself.
- We define the likelihood of a single training sample (ϕ, t) as its probability under the logistic regression model, as a function of the regression parameters $\mathbf{w}$ and w_0:

$$L_\phi(\mathbf{w}, w_0) = \begin{cases} y & \text{if } t = 1 \\ 1 - y & \text{if } t = 0 \end{cases} \qquad = y^t(1-y)^{1-t}$$

  where $y = y(\phi, \mathbf{w}, w_0)$.
- Assuming that the training samples are statistically independent, the likelihood of the training data $\{(\phi_n, t_n)\}_{n=1,\ldots,N}$ is

$$L(\mathbf{w}, w_0) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n}.$$

Logistic Regression

- It is computationally simpler to work in logarithmic space, thus we compute the log-likelihood

$$\tilde{L}(\mathbf{w}, w_0) = \sum_{n=1}^{N} t_n \ln y_n + (1 - t_n) \ln(1 - y_n).$$

- Note that always $\tilde{L}(\mathbf{w}, w_0) \leq 0$ (why?).
- We now define an error function as the negative log-likelihood

$$E(\mathbf{w}, w_0) = -\tilde{L}(\mathbf{w}, w_0) = -\sum_{n=1}^{N} t_n \ln y_n + (1 - t_n) \ln(1 - y_n).$$

- Our training criterion will be to *minimize* E, which is the same as *maximizing* L or $\tilde{L}$: the Maximum Likelihood criterion.

Logistic Regression

- Unfortunately, in contrast to "normal" linear regression, there is no closed-form solution to determine the parameter vector $\mathbf{w}$ and the bias w_0 for logistic regression.
- We will get to know a general way to solve such tasks during this course.
- During our lecture on probabilistic modeling, we will also reconsider the assumptions which underly Logistic Regression, and learn that under certain conditions it arises very naturally from imposing a basic probabilistic model.

Logistic Regression

- Unfortunately, in contrast to "normal" linear regression, there is no closed-form solution to determine the parameter vector $\mathbf{w}$ and the bias w_0 for logistic regression.
- We will get to know a general way to solve such tasks during this course.
- During our lecture on probabilistic modeling, we will also reconsider the assumptions which underly Logistic Regression, and learn that under certain conditions it arises very naturally from imposing a basic probabilistic model.
- If there is a decision boundary hyperplane which perfectly separates $\mathcal{C}_0$ and $\mathcal{C}_1$, the classes are said to be separable.
- In this case, the model is prone to *overfitting*: It will estimate the assignment probabilities badly.
- Even data points very close to the decision boundary will be assigned with high probability to their respective class.
- Again, probabilistic modeling offers a way to resolve this problem.

Classification: Summary

You have learned about

- nonparametric and parametric classification
- discriminant functions
- representation of a linear decision boundary in feature space
- Logistic regression as an example of a linear classifier.

USI/SUPSI
Istituto
Dalle Molle
di studi
sull'intelligenza
artificiale
IDSIA

Elements of the Foundations of Machine Learning

Foundations of Machine Learning

- We have gotten to know some elementary examples of Machine Learning algorithms.
- Before going on, we introduce some basic concepts which will help us to formalize what we have seen so far, and what we will get to know in the future.
- We start with the most basic question: How to formulate mathematically what learning means?
- You have already seen that it does *not* mean to memorize the training samples!

A Framework for Machine Learning

- Assume that our data is described by a probability distribution $P(x, t)$ over inputs and targets, where we assume both inputs and targets to be real-valued vectors: $x \in \mathbb{R}^M$, $t \in \mathbb{R}^N$.
- This key idea allows to express, in a mathematically sound way
  - ➔ the variability of data (even for a fixed class)
  - ➔ the ambigousness of the mapping between input data and target
  - ➔ and also our own lack of knowledge about this mapping!
- It also gives us a powerful set of tools to create practical algorithms.
- The probability calculus is at the heart of many important machine learning algorithms!

Bousquet: Introduction to Statistical Learning Theory. Machine Learning 2003, LNAI 3176, pp. 169–207, 2004.

A Framework for Machine Learning

- Assume a probability distribution $P(x, t)$ over inputs and targets.
- Let $y(x)$ be a predictor for t. Assuming any loss $L(t, y(x))$, define the risk as the expected loss over the entire probability space:

$$R(y) = E_{P(x,t)} L(t, y(x)) = \int L(t, y(x)) \mathrm{d}P(x, t).$$

- The goal of Machine Learning is now to find the predictor of t given x which minimizes the risk:

$$y^* = \underset{y:\mathbb{R}^M \to \mathbb{R}^N}{\arg\min}\ R(y) = \underset{y:\mathbb{R}^M \to \mathbb{R}^N}{\arg\min}\ E_{P(x,t)} L(t, y(x)).$$

- Note that by using this probabilistic framework, we avoid referring to specific data sets (like train and test dataset)!

Vapnik: Principles of Risk Minimization for Learning Theory. Proc. NIPS 1991.

The Irreducible Error

- Assume for a moment that we know the distribution $P(x, t)$.
- In some cases, we can directly obtain y^*, for example in the case of classification:

$$y^*(x) = \arg\max_t P(x, t)$$

- Q: Will the risk, i.e. the expected loss, be zero for the predictor y^*?

The Irreducible Error

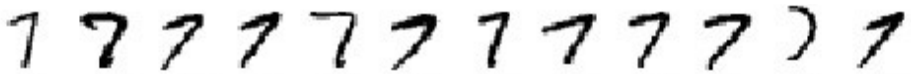
- Assume for a moment that we know the distribution $P(x, t)$.
- In some cases, we can directly obtain y^*, for example in the case of classification:

$$y^*(x) = \arg\max_t P(x, t)$$

- Q: Will the risk, i.e. the expected loss, be zero for the predictor y^*?
- **No**, because t might not be deterministic for a given x.
  - ➔ Only if x completely determines t, the risk of the optimal predictor is zero.
  - ➔ Mathematically, this means that for all x, $P(t|x) = 1$ for exactly one $t_{\text{True}}(x)$ and $P(t|x) = 0$ for all other t.

The Irreducible Error

- The prediction error which stems from the fact that the input does not completely determine the target is called the irreducible error.
- The irreducible error can be considered a form of noise affecting the inputs, the targets, or both. It is *independent* of any concrete method which we use for the prediction of targets.
- In practice, such noise can derive from inexact measurements of data, from unaccounted sources of variation, and many other factors.
- In the case of classification, we (usually) map a continuous input to discrete output, so there will be border regions with unclear class assignments.
- As an example, consider a subset of the famous *MNIST* corpus of handwritten digits. Do you think all these examples are clearly the digit "7"?



A Framework for Machine Learning

- Clearly, we do *not* know the true distribution of the data $P(x, t)$.
- How can we estimate (and subsequently minimize) the risk?

A Framework for Machine Learning

- Clearly, we do *not* know the true distribution of the data $P(x, t)$.
- How can we estimate (and subsequently minimize) the risk?
- We are given a finite set of observations $\mathcal{D} = \{(x_n, t_n)\}_{n=1,\ldots,N}$.
- Idea: Approximate expectations over P by averaging over the data, specifically:

$$R(y) = E_{P(x,t)} L(t, y(x)) \approx \frac{1}{N} \sum_{n=1}^{N} L(t_n, y(x_n)) =: R_{\text{emp}}(y).$$

- $R_{\text{emp}}(y)$ is called empirical risk. The observations $\mathcal{D}$ are usually called training data.

A Framework for Machine Learning

- The entire field of Machine Learning deals with *minimizing the true risk, based on estimations using the empirical risk*.
- We will get to know a large variety of algorithms and methods where we give the predictor $y(x)$ a specific mathematical form, allowing minimization of the empirical risk.
- However, does the empirical risk always reflect the true risk?

A Framework for Machine Learning

- The entire field of Machine Learning deals with *minimizing the true risk, based on estimations using the empirical risk*.
- We will get to know a large variety of algorithms and methods where we give the predictor $y(x)$ a specific mathematical form, allowing minimization of the empirical risk.
- However, does the empirical risk always reflect the true risk?
- In general, **no**!
  - ➔ Remember that we allow *any* function from the input space to the target space as a predictor.
  - ➔ For example, a function which correctly maps all training data points and is random on all other points achieves zero empirical risk, but high true risk.
- We want to avoid such pathological cases. Which functions are "reasonable" as predictors?

A Framework for Machine Learning

- A function like the one on the last slide has a high complexity, whereas a *structured representation* of the data should have a much simpler form.
- Thus, we restrict the predictors $y(x)$ which we take into account to a subset of the functions from $\mathbb{R}^M$ to $\mathbb{R}^N$: the hypothesis class $\mathcal{H}$.
- This generalizes and formalizes the concept of regularization which we covered earlier!
- We hope that this helps us to obtain a predictor whose empirical risk (on the training data) reflects the true risk (on unseen test data).
- Clearly, this is not automatically true...

A Framework for Machine Learning

- We now define the optimal predictor *within the hypothesis class* $\mathcal{H}$:

$$y^*_{\mathcal{H}} = \arg\min_{y \in \mathcal{H}} R(y) = \arg\min_{y \in \mathcal{H}} E_{P(x,t)} L(t, y(x))$$

as well as the empirically optimal predictor given a dataset $\mathcal{D} = \{(x_n, t_n)\}_{n=1,\ldots,N}$:

$$y^{emp}_{\mathcal{H}} = \arg\min_{y \in \mathcal{H}} R_{\text{emp}}(y) = \arg\min_{y \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} L(t_n, y(x_n)).$$

- In almost all cases, exactly computing the optimal predictor in $\mathcal{H}$ is intractable, and it needs to be approximated.
- *All Machine Learning algorithms aim at optimally defining the hypothesis class* $\mathcal{H}$*,* such that efficient optimization over a wide range of functions is possible, while retaining generalization ability.

A Framework for Machine Learning

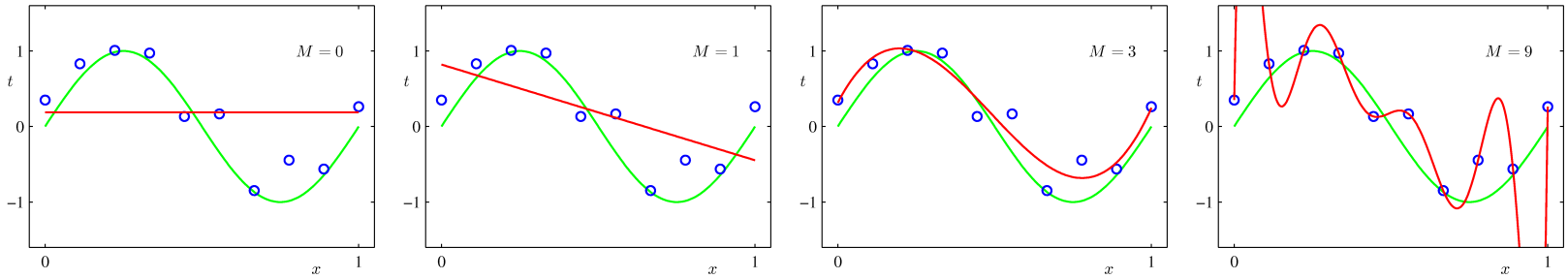
- Based on the probabilistic framework, many bounds on the difference between empirical and true risk have been proposed, typically of the form

$$\forall y \in \mathcal{H} \qquad R(y) - R_{\text{emp}}(y) < O\left(\sqrt{\frac{\text{cap}(\mathcal{H})}{N}}\right)$$

  where N is the number of samples, and $\text{cap}(\mathcal{H})$ measures the "capacity" or "richness" of the function class $\mathcal{H}$ (not covered in this course).
- Note that the true risk is typically assumed to be greater than the empirical risk.
- We see that it is necessary to balance the capacity of the hypothesis class and the number of data samples:
  - ➔ If there are not enough elements in $\mathcal{H}$, even the optimal predictor might not be very good.
  - ➔ If there are too many elements in $\mathcal{H}$, the empirically best predictor might be far away from the true optimal predictor.

Definitions

- A predictor whose true risk is close to the empirical risk on the training data is said to generalize well.
- A predictor whose empirical risk is low, but whose true risk is high, is said to overfit to the training data.
- A predictor whose empirical risk is high is said to underfit the data.
- Which of the predictors in the figure overfit, and which ones underfit?



Img src: Bishop, figure 1.4

Using Data Sets

- We tune our Machine Learning system on the training data, obtaining the empirical risk. But how can we now estimate the true risk?
- This requires a *separate* dataset which is *not* used for tuning the system, but only for measuring the loss after training: the test data.
- In practical scenarios, even this might not guarantee a good estimate of the true risk: If we perform system tuning many times with slightly different metaparameters, always reusing the same test data, we implicitly tune the system to this test data.
- Good practice requires to use *three* different data sets:
  - ➔ The training data is used for tuning the system.
  - ➔ The validation data is used to determine the quality of the trained system and to guide further optimization steps.
  - ➔ The test data is used *only when the system is completely tuned* to determine the final quality (e.g. for writing a final report or similar).

Foundations: Summary

In this section, you have gotten to know

- a mathematical formulation of the task of Machine Learning
- generalization, overfitting, and underfitting revisited
- splitting your data set to obtain statistically valid results.