

# Machine learning foundations

Mitko Veta

Eindhoven University of Technology  
Department of Biomedical Engineering

2024

# Learning goals

At the end of this lecture you will:

- ▶ Have an understanding of the goal of machine learning (ML) models.
- ▶ \*Have a good understanding of basic mathematical concepts used in ML and be able to apply them in the design and implementation of ML methods.
- ▶ Have a good understanding of the basic principles of machine learning (ML) and be able to apply them in the analysis of ML methods.
- ▶ Be able to design good experimental setups for developing ML models.
- ▶ \*Have a good understanding of the different evaluation measures for ML models.

\* Covered in video lectures

# Overview

Topics covered in this lecture:

1. Linear algebra
2. Gradient of a function
3. Two simple machine learning models
  - Linear model
  - Nearest-neighbours model
4. Probability theory
5. Model capacity, underfitting and overfitting
6. Model selection
7. Bias and variance trade-off
8. Maximum likelihood estimation
9. Model evaluation
10. Supervised and unsupervised learning algorithms
11. Ensembling

# Note on the slides

This set of slides is larger than the one used during the lectures. It includes some additional material that you can use as a guide when studying.

# Linear algebra

Materials:

- ▶ Chapter 1.2 from Goodfellow et al., *Deep Learning*
- ▶ Kolter et al., “[Linear Algebra Review and Reference](#)”

# Scalars

- ▶ A scalar is a single number (integer, real, rational, ...).
- ▶ Denoted by italics  $a, n, x$

# Vectors

- ▶ A vector is a 1-D array of numbers (integer, real, rational, ...)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

- ▶ Example notation for type and size  
 $\mathbf{x} \in \mathbb{R}^n$

# Matrices

- ▶ A matrix is a 2-D array of numbers

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}$$

- ▶ Example notation for type and shape  
 $\mathbf{A} \in \mathbb{R}^{m \times n}$



# Tensors

- ▶ A tensor is an array of numbers that may have
  - ▶ a zero dimensions and be a scalar,
  - ▶ one dimension and be a vector,
  - ▶ two dimensions and be a matrix,
  - ▶ more dimensions ...

**Side note:** One of the most popular frameworks for implementing deep machine learning models is called TensorFlow (<https://www.tensorflow.org/>).

# Transpose matrix

$$(\mathbf{A}^T)_{i,j} = \mathbf{A}_{j,i}$$

$$\mathbf{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

The transpose matrix is a mirror image with regard to the main diagonal

$$(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$$

# Identity matrix

- ▶ Identity matrix  $I_3$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- ▶ The identity matrices are neutral elements in matrix-matrix and matrix-vector multiplication, e.g.

$$\forall \mathbf{x} \in \mathbb{R}^n : I_n \mathbf{x} = \mathbf{x} I_n = \mathbf{x}$$

## Matrix (dot) product

$$\mathbf{C} = \mathbf{AB}$$

The matrices must be compatible: an  $m \times n$  matrix is multiplied with an  $n \times r$  matrix and as a result an  $m \times r$  matrix is obtained

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

$$\mathbf{A} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \times \mathbf{B} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix} = \mathbf{C} [4 \times 5]$$

$$C_{2,5} = A_{2,1}B_{1,5} + A_{2,2}B_{2,5} + A_{2,3}B_{3,5} = 4 \cdot 5 + 5 \cdot 10 + 6 \cdot 15 = 160$$

# Matrix (dot) product

- ▶ In general matrix multiplication is not commutative, i.e., most of the time  $\mathbf{AB} \neq \mathbf{BA}$ .
- ▶ Depending on the dimensions sometimes  $\mathbf{AB}$  or  $\mathbf{BA}$  are not possible.
- ▶ As a special case the matrix can be a (column or row) vector; an  $m \times n$  matrix is multiplied with a  $n \times 1$  vector to obtain a  $m \times 1$  vector.

# Systems of linear equations

$$A_{1,1}x_1 + A_{1,2}x_2 + \dots + A_{1,n}x_n = b_1$$

$$A_{2,1}x_1 + A_{2,2}x_2 + \dots + A_{2,n}x_n = b_2$$

...

$$A_{m,1}x_1 + A_{m,2}x_2 + \dots + A_{m,n}x_n = b_m$$

- ▶  $A_{*,*}$  and  $b_*$  are the knowns,  $x_*$  are the unknowns.
- ▶ In matrix form:  **$A\mathbf{x} = \mathbf{b}$**

# Systems of linear equations

►  $\mathbf{Ax} = \mathbf{b}$  expands to

$$\mathbf{A}_{1,:}\mathbf{x}_1 = \mathbf{b}_1$$

$$\mathbf{A}_{2,:}\mathbf{x}_2 = \mathbf{b}_2$$

...

$$\mathbf{A}_{m,:}\mathbf{x}_m = \mathbf{b}_m$$

# Solving systems of linear equations

- ▶ A linear system of equations can have
  - ▶ no solutions,
  - ▶ many solutions,
  - ▶ exactly one solution.
- ▶ Only one solution implies that multiplication by a matrix is an invertible operation.



# Matrix inversion

- ▶ Matrix inverse is defined with

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$$

- ▶ A system of linear equations can be solved using inverse matrix

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

$$\mathbf{I}_n\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

- ▶ This is useful mostly for abstract analysis.
- ▶ From a numerical point of view there are much more efficient methods.

# Invertibility

A matrix cannot be inverted if

- ▶ the number of rows and columns is not the same, or
- ▶ some rows and columns are "redundant" ("linearly dependent", "low rank").

# Moore-Penrose pseudoinverse

- ▶ Matrix inversion is not defined on matrices that are not square.
- ▶ The **Moore-Penrose pseudoinverse** is defined as

$$\mathbf{A}^+ = \lim_{\alpha \searrow 0} (\mathbf{A}^T \mathbf{A} + \alpha \mathbf{I})^{-1} \mathbf{A}^T$$

# Moore-Penrose pseudoinverse

Now we can consider

$$\mathbf{x} = \mathbf{A}^+ \mathbf{y}$$

- ▶ If the equation has
  - ▶ exactly one solution: this is the same as inverse,
  - ▶ no solution: gives the solution with the smallest error,  $\|\mathbf{Ax} - \mathbf{y}\|_2$
  - ▶ many solutions: gives the solution with the smallest norm of  $\mathbf{x}$ .

# Singular value decomposition

- ▶ Similar to eigenvalue decomposition
- ▶ More general: matrix need not be square

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

- ▶  $\mathbf{U}$  and  $\mathbf{V}$  are square matrices and are both orthogonal,  $\mathbf{D}$  is diagonal.
- ▶ The diagonal elements of  $\mathbf{D}$  are called **singular values** of matrix  $\mathbf{A}$ ; the columns of  $\mathbf{U}$  and  $\mathbf{V}$  are **left-singular** and **right-singular vectors** of  $\mathbf{A}$ , respectively.

# Computing the pseudoinverse

- ▶ Efficient implementations are based on the formula allowed by the singular decomposition

$$\mathbf{A}^+ = \mathbf{V}\mathbf{D}^+ \mathbf{U}^T$$

- ▶  $\mathbf{U}$ ,  $\mathbf{D}$ ,  $\mathbf{V}$  are from the singular value decomposition of  $\mathbf{A}$ .
- ▶ The pseudoinverse  $\mathbf{D}^+$  of  $\mathbf{D}$  is obtained by taking the reciprocal non-zero elements and after that taking the transpose of the resulting matrix.

# Norms

- ▶ Norms are functions that measure how "large" a vector is.
- ▶ Similar to a distance between zero and the point represented by the vector
  - ▶  $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = 0$
  - ▶  $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$  (**the triangle inequality**)
  - ▶  $\forall \alpha \in \mathbb{R} : f(\alpha \mathbf{x}) = \alpha f(\mathbf{x})$

# Norms

- ▶  $L^p$ - norm

$$\|\mathbf{x}\|_p = \left( \sum_i |x_i|^p \right)^{\frac{1}{p}}$$

- ▶ Most popular  $L^2$ -norm (for  $p = 2$ )
- ▶  $L_1$ -norm (for  $p = 1$ ):  $\|\mathbf{x}\|_1 = \sum_i |x_i|$
- ▶ Max norm (for infinite  $p$ ):  $\|\mathbf{x}\|_\infty = \max_i |x_i|$



# Special vectors and matrices

- ▶ Unit vector  $\|\mathbf{x}\|_n = 1$
- ▶ Symmetric matrix  $\mathbf{A} = \mathbf{A}^T$
- ▶ Orthogonal matrix

$$\mathbf{A}\mathbf{A}^T = \mathbf{I} = \mathbf{A}^T\mathbf{A}$$

- ▶ It follows that for orthogonal matrices  $\mathbf{A}^T = \mathbf{A}^{-1}$

# Eigendecomposition

- ▶ Eigenvector and eigenvalue

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

- ▶ Eigendecomposition of a matrix

$$\mathbf{A} = \mathbf{V}\text{diag}(\lambda)\mathbf{V}^{-1}$$

where  $\text{diag}(\lambda)$  is a diagonal matrix having the (scalar) eigenvalues  $\lambda$  as diagonal elements.

# Eigendecomposition

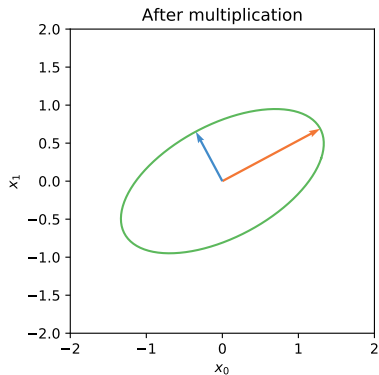
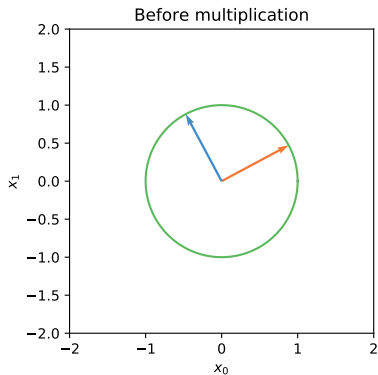
- ▶ Every real symmetric matrix has a real orthogonal eigendecomposition

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$$

where  $\mathbf{Q}$  is an orthogonal matrix composed of eigenvectors of  $\mathbf{A}$  and  $\mathbf{\Lambda}$  is a diagonal matrix.

- ▶ The eigenvalue  $\Lambda_{ii}$  is associated with the eigenvector in column  $i$  of  $\mathbf{Q}$ , denoted as  $\mathbf{Q}_{:,i}$ .
- ▶ We can think of  $\mathbf{A}$  as scaling space by factor  $\lambda_i$  in the direction of its corresponding eigenvector  $\mathbf{v}^{(i)}$  (represented by  $\mathbf{Q}_{:,i}$ ).

# Effect of eigenvalues



# Eigendecomposition

- ▶ From the eigendecomposition we learn useful properties of the matrix.
- ▶ The eigendecomposition of a real symmetric matrix is used in optimization of quadratic expressions of the form  $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$  under the constraint  $\|\mathbf{x}\|_2 = 1$ .
- ▶ For instance, if  $\mathbf{x} = \mathbf{v}^{(i)}$ , then  $f(\mathbf{x}) = \lambda_i$ , when  $\mathbf{v}^{(i)}$  is an eigenvector of  $A$  and  $\lambda_i$  is its corresponding eigenvalue.
- ▶ The maximal (minimal) value of  $f$  within the constraint region is equal to the maximal (minimal) eigenvalue.

- ▶ A **trace** of a matrix is defined as

$$Tr(\mathbf{A}) = \sum_i \mathbf{A}_{i,i}$$

- ▶ Expressions in terms of the trace operators allow to exploit many useful identities, e.g.

$$Tr(\mathbf{ABC}) = Tr(\mathbf{BCA}) = Tr(\mathbf{CAB})$$

# Prelude: Gradient of a function

Materials:

- ▶ Chapters 1.4 and 1.5 from Goodfellow et al., *Deep Learning*
- ▶ Kolter et al., “*Linear Algebra Review and Reference*”

# Gradient of a function

- ▶ Let  $f : \mathbb{R}^{m \times n} \mapsto \mathbb{R}$  be a function that takes  $m \times n$  matrix  $\mathbf{A}$  as input and returns a real number (scalar).
- ▶ A **gradient** of  $f$  with respect to  $A$  is the matrix

$$\nabla_{\mathbf{A}} f(\mathbf{A}) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \frac{\partial f}{\partial A_{12}} & \cdots & \frac{\partial f}{\partial A_{1n}} \\ \frac{\partial f}{\partial A_{21}} & \frac{\partial f}{\partial A_{22}} & \cdots & \frac{\partial f}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \frac{\partial f}{\partial A_{m2}} & \cdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

- ▶ i.e. an  $m \times n$  matrix with

$$(\nabla_{\mathbf{A}} f(\mathbf{A}))_{ij} = \frac{\partial f}{\partial A_{ij}}$$

- ▶ The size of the gradient of  $\mathbf{A}$  is the same as the size of  $A$ .



# Gradient

- ▶ In the special (but more common) case when  $A$  is a vector we obtain the gradient

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{bmatrix}$$

- ▶ In general to define a gradient we require that the function returns a **real** value.

# Jacobian

- ▶ The Jacobian  $\mathbf{J}_f$  is a generalization of the gradient for vector valued functions.
- ▶ Let  $\mathbf{f} : \mathbb{R}^n \mapsto \mathbb{R}^m$  be a function that takes  $n$ -dimensional vector  $\mathbf{x}$  as input and returns a  $m$ -dimensional vector as an output.
- ▶ The Jacobian  $\mathbf{J}_f$  is defined as

$$\mathbf{J}_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

- ▶ Note that for the special case of a scalar-valued function, the Jacobian is the transpose of the gradient.

# Optimization

- ▶ Most machine learning methods involve some kind of optimization.
  - ▶ One exception is the  $k$ -Nearest neighbour classifier introduced later.
- ▶ Optimization means minimizing or maximizing some function  $f(\mathbf{x})$ , i.e. finding the values of  $\mathbf{x}$  for which  $f(\mathbf{x})$  has a minimum or a maximum.
- ▶ Notation:  $\mathbf{x}^* = \operatorname{argmin} f(\mathbf{x})$

# Gradient-based optimization

- ▶ The derivative tells us how to change  $x$  in order to make a small improvement of  $f(x)$ .
- ▶ Therefore, derivatives can be useful in optimization.

# Two simple machine learning models

Materials:

- ▶ Chapter 2.3 from Friedman et al., *The Elements of Statistical Learning*

## Some notations

- ▶ We denote an input variable with the symbol  $x$  (scalar) or  $\mathbf{x}$  (vector).
- ▶ The  $i$ -th component of a vector input  $\mathbf{x}$  is denoted as  $x_i$ .
- ▶ Quantitative (numerical) outputs are denoted with  $y$ .
- ▶ Qualitative outputs are denoted with  $g$  (from group) and take values from a set  $\mathcal{G}$ .
- ▶ Matrices are denoted with bold and uppercase letters  $\mathbf{X}$  for instance, a set of  $N$  input  $p$ -vectors  $\mathbf{x}_i$  ( $1 \leq i \leq N$ ) is "packed" in a  $N \times p$  input matrix  $\mathbf{X}$ .
- ▶ Since by default vectors are assumed to be column vectors, the rows of  $\mathbf{X}$  are the transposes  $\mathbf{x}_i^T$ .

# The learning task

- ▶ Given a value of the input vector  $\mathbf{x}$  make a good prediction of the output  $y$ , denoted as  $\hat{y}$ .
- ▶ Both  $y$  and  $\hat{y}$  should take values from the same numerical set.
- ▶ Similarly,  $g$  and  $\hat{g}$  should both take values from the same set  $\mathcal{G}$ .
- ▶ We suppose that we have available a set of measurements  $(\mathbf{x}_i, y_i)$  or  $(\mathbf{x}_i, g_i)$  ( $1 \leq i \leq N$ ) called **training data** (in matrix form:  $(\mathbf{X}, \mathbf{y})$  and/or  $(\mathbf{X}, \mathbf{g})$ ).
- ▶ Our task is to construct a prediction rule based on the training data.

# The learning task

Example:

- ▶ **Variable values:** Let  $g$  (and therefore also  $\hat{g}$ ) be two valued (categorical), e.g.  $\mathcal{G} = \{\text{BLUE}, \text{ORANGE}\}$ .
- ▶ **Encoding of  $g$ s with  $y$ s:** Then each class can be encoded binary, i.e., with  $y \in \{0, 1\}$ , e.g., **BLUE** and **ORANGE**, would correspond to 0 and 1, respectively.
- ▶ **Predicted output values:**  $\hat{y}$  ranges over the interval  $[-\infty, +\infty]$  (of which  $\{0, 1\}$  is a subset).
- ▶ **Prediction rule:**  $\hat{g}$  is assigned a (class label) **BLUE** if  $\hat{y} < 0.5$  and **ORANGE**, otherwise.



# Two simple approaches to prediction

- ▶ Linear model fit
  - ▶ strong assumptions about the structure of the decision boundary
- ▶  $k$ -nearest neighbours
  - ▶ weak assumptions about the structure of the decision boundary

# Linear model fit by least squares

- ▶ Despite relative simplicity one of the most important statistical tools
- ▶ Input vector  $\mathbf{x}^T = (x_1, x_2, \dots, x_p)$
- ▶ Output  $y$  predicted using the model

$$\hat{y} = \hat{w}_0 + \sum_{j=1}^p x_j \hat{w}_j$$

- ▶  $\hat{w}_i$  ( $0 \leq i \leq p$ ) are the parameters of the linear model
- ▶ In vector form

$$\hat{y} = \hat{\mathbf{w}}^T \mathbf{x} = \mathbf{x}^T \hat{\mathbf{w}}$$

using the fact that the scalar (inner) product of two vectors is a commutative operation.

# Linear model fit by least squares

- ▶ We assume that  $w_0$  is in  $\mathbf{w}$  and 1 is included in  $\mathbf{x}$ .
- ▶  $\hat{y}$  is a scalar, but in general can be a  $k$ -vector  $\hat{\mathbf{y}}$ , in which case  $\mathbf{w}$  becomes a  $p \times k$  matrix of coefficients.

# Linear model fit by least squares

Some hyper(space) terminology:

- ▶ Points  $\mathbf{x}, \hat{y}$  form a **hyperplane** in the  $(p + 1)$ -dimensional input-output hyperspace.
- ▶ If  $\mathbf{x}$  is extended with constant 1 then the hyperplane includes the origin and it forms a **subspace**.
- ▶ If 1 is not included then the hyperplane is an **affine** set and it cuts the  $y$ -axis at the point  $(\mathbf{0}, \hat{w}_0)$ , where the vector  $\mathbf{0}$  has all  $x_i$  coordinates equal to 0.
- ▶ Reminder: from now on we assume that 1 is included in  $\mathbf{x}$  and  $\hat{w}_0$  in  $\hat{\mathbf{w}}$ .
- ▶ The function  $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  defined on the  $p$ -dimensional (input) space is a **linear** function (we omit the hats over the  $\mathbf{w}$ s since now we consider them as free variables).
- ▶ The gradient  $\nabla f(\mathbf{x})$  is a vector pointing along the direction of maximal change.

# Linear model fit by least squares

- ▶ There are many ways to fit a linear model to a training dataset.
- ▶ **Least squares** method
  - ▶ We need to find coefficients  $\hat{w}_i$  which minimize the error estimated with the **residual sum of squares**

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^N (y_i - \mathbf{x}_i^T \mathbf{w})^2$$

assuming  $N$  input-output pairs.

- ▶  $\text{RSS}(\mathbf{w})$  is a quadratic function.
- ▶ A minimum always exists though not necessarily a unique one.

# Linear model fit by least squares

- ▶ We look for the solution  $\hat{\mathbf{w}}$  using the matrix notation:
- ▶  $\mathbf{y} = [y_1, y_2, \dots, y_N]^T$  is the vector formed from the  $N$  output vectors and  $\mathbf{X}$  is an  $N \times p$  matrix

$$\text{RSS}(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

- ▶ To find the minimum we differentiate with respect to  $\mathbf{w}$  which gives

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

For details about the derivation check equations 4 and 5 in [this document](#).

# Linear model fit by least squares

- ▶ To find the minimum our derivative must be **0**, hence:

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = \mathbf{0}$$

$$\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{0}$$

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w}$$

- ▶ If  $\mathbf{X}^T\mathbf{X}$  is non-singular there exists a unique solution given by

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

## Discussion point 1

Why we cannot simply solve for  $\hat{\mathbf{w}}$  in the following way?

$$\mathbf{y} - \mathbf{X}\mathbf{w} = \mathbf{0}$$

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

$$\hat{\mathbf{w}} = \mathbf{X}^{-1}\mathbf{y}$$



# Linear model fit by least squares

- ▶ For each input  $\mathbf{x}_i$  there corresponds the fitted output

$$\hat{y}_i = \hat{y}_i(\mathbf{x}_i) = \hat{\mathbf{w}}^T \mathbf{x}_i$$

.

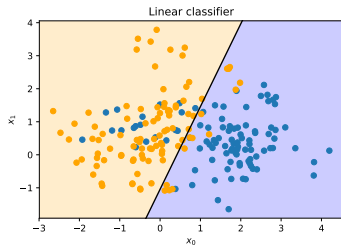
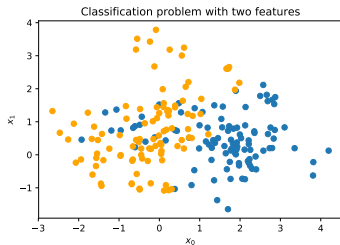
- ▶ This is called “making a prediction” for  $\mathbf{x}_i$ .
- ▶ The entire fitted surface (hyperplane) is fully characterized by the parameter vector  $\hat{\mathbf{w}}$ .
- ▶ After fitting the model, we can “discard” the training dataset.

## Example: Linear model fit by least squares

- ▶ Scatter plot (on next slide) of training data on a pair of inputs  $x_1$  and  $x_2$
- ▶ Output class variable  $g$  has two values BLUE and ORANGE.
- ▶ Linear regression model fitted with the response variable  $y$  coded as 0 for BLUE and 1 for ORANGE.
- ▶ Fitted values  $\hat{y}$  converted to a fitted class variable  $\hat{g}$  as

$$\hat{g} = \begin{cases} \text{BLUE} & \text{if } \hat{y} \leq 0.5 \\ \text{ORANGE} & \text{if } \hat{y} > 0.5 \end{cases}$$

# Example: Linear model fit by least squares



## Example: Linear model fit by least squares

- ▶ Two classes separated in the plane ( $\mathbb{R}^2$ ) by the decision boundary  $\{\mathbf{x} : \mathbf{w}^T \mathbf{x} = 0.5\}$
- ▶  $\{\mathbf{x} : \mathbf{w}^T \mathbf{x} < 0.5\}$  set of BLUE points
- ▶  $\{\mathbf{x} : \mathbf{w}^T \mathbf{x} \geq 0.5\}$  set of ORANGE points

## Example: Linear model fit by least squares

- ▶ Wrong classifications on both sides of the boundary
- ▶ Are the errors caused by the model or are they unavoidable?
- ▶ Two possible scenarios
  - ▶ **Scenario 1:** data generated from bivariate Gaussian distribution
  - ▶ **Scenario 2:** data generated from 10 Gaussian distributions; the means of these distributions are also distributed as Gaussian
- ▶ In Scenario 1 the linear boundary is the best we can do since the overlap is inevitable.
- ▶ In Scenario 2 the linear boundary is unlikely to be optimal (in fact the boundary is non-linear and disjoint).

# Nearest-neighbours model

- ▶ In nearest-neighbour methods  $\hat{y}(\mathbf{x})$  is determined based on the inputs (points) in the training set  $\mathcal{T}$  which are "closest" to the input  $\mathbf{x}$ .
- ▶  $k$ -nearest neighbour fit is defined as

$$\hat{y}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i$$

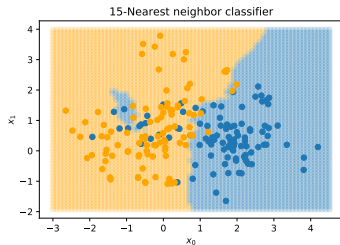
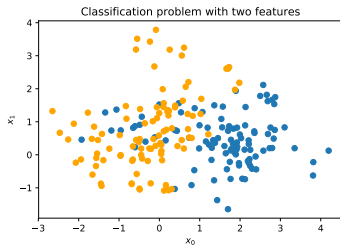
where  $N_k(\mathbf{x})$  is the neighbourhood of  $\mathbf{x}$  consisting of the  $k$  "closest" points to  $\mathbf{x}$ .

- ▶ "Closeness" requires a definition of **metrics**.
- ▶ For the moment we assume Euclidian distance (each  $\mathbf{x}$  is a point in the hyperspace).
- ▶ An average of the classes of the  $k$  closest points (but only for binary classification problem).

## Back to the BLUE and ORANGE example

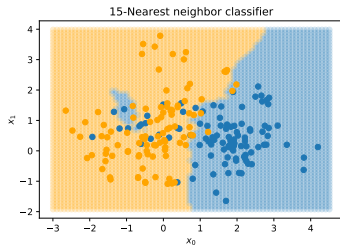
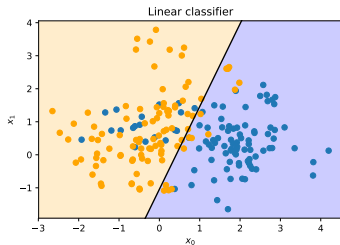
- ▶ We use the same training data as in the linear model example.
- ▶ New borderline between the classes generated with 15-nearest-neighbour model.
- ▶ Since ORANGE is encoded as 1  $\hat{y}$  is the proportion of ORANGE points in the 15-neighbourhood.
- ▶ Class ORANGE assigned to  $\mathbf{x}$  if  $\hat{y}(\mathbf{x}) > 0.5$  (majority is ORANGE).

# 15-Nearest neighbour classifier

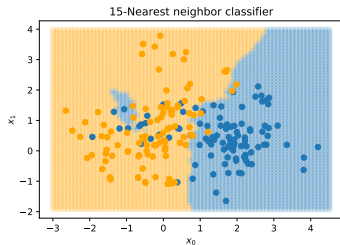
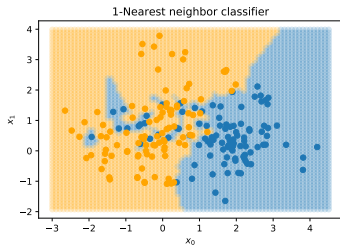




# Linear classifier vs. 15-Nearest neighbour



# 1-Nearest neighbour vs. 15-Nearest neighbour



# Comparison of the techniques

- ▶ 15-NN seems to work better than the linear classifier since fewer points are missclassified.
- ▶ On the other hand, **none** of the points in the 1-NN case was misclassified!?
- ▶ Actually with the 1-NN method the error on **training data** is always 0.
- ▶ An independent test set needed to obtain a better comparison of the methods.

# Comparison of techniques

- ▶ At first sight it looks like k-NN has only one parameter,  $k$  versus  $p$  parameters (number of weights  $w_i$ ) of the linear model.
- ▶ The **effective** number of parameters of k-NN is  $N/k$  which is in general bigger than  $p$  ( $N$  is the size of the training set).
- ▶ For instance, assume non-overlapping neighbourhoods
  - ▶ There will be  $N/k$  neighbourhoods.
  - ▶ To each neighbourhood there correspond one parameter (the mean of the elements of the neighbourhood).

## Discussion point 2

Assume that you are building a machine learning model to be used as an *aid* by clinicians for decision making. The inputs to the model are a number of *biomarkers* describing the condition of the patient.

The clinician specifies that they are interested in a model that is *interpretable*, i.e. a model that will not only output a prediction but also give an indication about which biomarkers are important when making the prediction.

You can either use a linear model or a  $k$ -NN classifier. What is the better choice in your opinion?

# Probability theory

Materials:

- ▶ Chapter 1.3 from Goodfellow et al., *Deep Learning*

# Probability theory

- ▶ Probability theory is a mathematical framework for dealing with uncertainty, i.e., modeling and analyzing uncertain events and statements
- ▶ In AI probability theory is used in two major ways:
  - ▶ To design AI systems, i.e., derive models and expressions and the corresponding algorithms.
  - ▶ To analyze the behaviour of the AI systems.

# Probability theory

- ▶ A **random variable** is a variable that can take values randomly.
- ▶ We will denote random variables with plain (ordinary text) typeface and their values with standard math typeface for example, if the random variable is denoted as  $x$  its values can be  $x_1$  and  $x_2$ .
- ▶ A vector-valued random variable is denoted with bold typeface, e.g.  $\mathbf{x}$ .
- ▶ On its own a random variable just denotes the set of its possible values; to get its full meaning it needs to be coupled with a distribution.



# Probability theory

- ▶ There are two types of random variables: **discrete** and **continuous**.
- ▶ Consequently there are two ways to describe probability distributions: **probability mass functions** and **probability density functions**.

# Probability mass function

- ▶ The domain of a probability mass function  $P$  is the set of all possible states of the random variable  $x$ .
- ▶  $\forall x \in \mathcal{X} : 0 \leq P(x) \leq 1$ 
  - ▶ An impossible event has probability 0 and no state can be less probable than that.
  - ▶ An event that is guaranteed to happen has probability 1 and no state can have a greater chance of occurring.
- ▶  $\sum_{x \in \mathcal{X}} P(x) = 1$ 
  - ▶ We say that  $x$  is **normalized**.
- ▶ Example: Uniform distribution:  $P(x = x_i) = \frac{1}{k}$ .

# Probability density function

- ▶ The domain of the probability density function  $p$  must be the set of all possible states of  $x$ .
- ▶  $\forall x \in x : p(x) \geq 0$ .



$$\int p(x) dx = 1$$

- ▶ Example: uniform distribution  $u(x; a, b) = \frac{1}{b-a}$ , for  $x \in [a, b]$

# Conditional probability

- ▶ **Conditional probability** is the probability of some event provided that some other event has happened.
- ▶ Given two random variables  $x$  and  $y$ , the conditional probability that  $y$  has value  $y$  provided that we know that  $x$  has value  $x$  is given by

$$P(y = y \mid x = x) = \frac{P(x,y)}{P(x = x)}$$

- ▶ Another way to see this formula is

$$P(x,y) = P(x = x)P(y = y \mid x = x)$$

i.e., the probability of  $x$  and  $y$  occurring together is equal to the probability of occurrence of  $x$  times the probability of  $y$  occurring provided  $x$  has occurred.

# Expectation

- ▶ The **expectation** or **expected** value of a function  $f(x)$  with respect to a probability distribution  $P(x)$  is the average value of  $f$  over all values  $x$  assuming they are drawn from  $P$



$$\mathbb{E}_{x \sim P}[f(x)] = \sum_x P(x)f(x)$$



$$\mathbb{E}_{x \sim P}[f(x)] = \int p(x)f(x)dx$$

- ▶ Linearity of expectations:

$$\mathbb{E}_x[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}_x[f(x)] + \beta \mathbb{E}_x[g(x)]$$

# Variance and covariance

- ▶ The **variance** gives a measure of variation of the values of a random variable  $x$

$$\text{Var}(f(x)) = \mathbb{E}[(f(x) - E[f(x)])^2]$$

Square root of the variance is called **standard deviation**.

- ▶ The **covariance** is a measure of linear relation as well as scale between

$$\text{Cov}(f(x), g(x)) = \mathbb{E}[(f(x) - E[f(x)])(g(x) - E[g(x)])]$$

# Covariance matrix

- ▶ The **covariance matrix** of a random vector  $\mathbf{x} \in \mathbb{R}^n$  is a  $n \times n$  matrix with elements

$$\text{Cov}(\mathbf{x})_{i,j} = \text{Cov}(x_i, x_j)$$

- ▶ The diagonal elements of the matrix give the variance

$$\text{Cov}(x_i, x_i) = \text{Var}(x_i)$$

# Bernouli Distribution

- ▶ A distribution over a single binary random variable
- ▶ Controlled by a single parameter  $\phi \in [0, 1]$  which corresponds to the probability of the random variable taking the value 1
- ▶ Properties:

$$P(x = 1) = \phi$$

$$P(x = 0) = 1 - \phi$$

$$P(x = x) = \phi^x (1 - \phi)^{1-x}$$

$$\mathbb{E}_x[x] = \phi$$

$$\text{Var}(x) = \phi(1 - \phi)$$



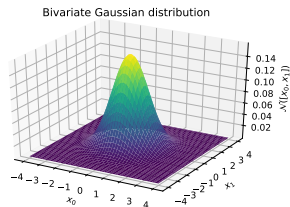
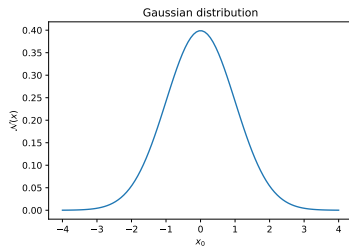
# Gaussian distribution

- ▶ The most commonly used distribution, also called **normal distribution**.
- ▶ Controlled by two parameters  $\mu \in \mathbb{R}$  (the **mean**) and  $\sigma \in (0, \infty)$ , (the **standard deviation**)

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right)$$

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sqrt{\frac{1}{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

# Gaussian distribution

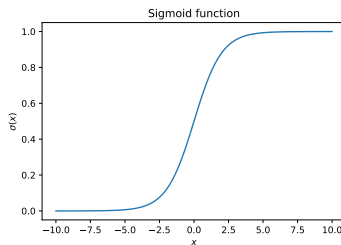


# Logistic sigmoid

- ▶ A useful function that we are going to consider

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

- ▶ The Logistic (sigmoid) function is commonly used to parametrize Bernoulli distributions.



# Bayes' rule

- ▶ Suppose know  $P(y | x)$ , but we actually need  $P(x | y)$ . If we know  $P(x)$  then we can compute

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

Although it appears in the formula prior knowledge  $P(y)$  is not needed since usually it can be computed as  $\sum_x P(y | x)P(x)$

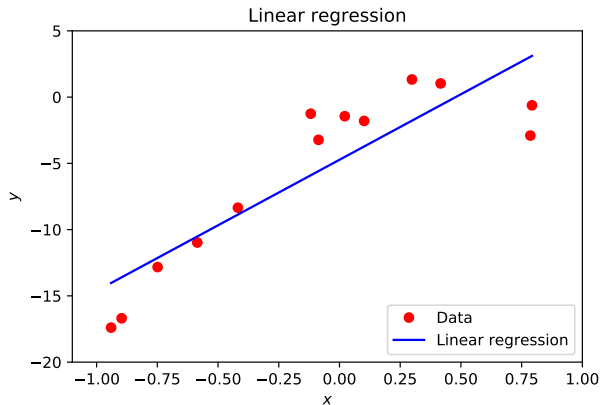
- ▶ It can be straightforwardly derived from the conditional probability formula.
- ▶ It could have be named also after Laplace who independently found it, generalized it, and introduced it in practice.

# Model capacity, underfitting and overfitting

Materials:

- ▶ Chapter 1.5.2 from Goodfellow et al., *Deep Learning*

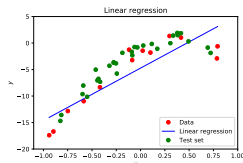
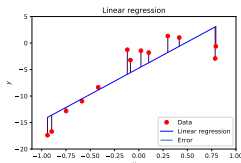
# Linear regression



$$\hat{y} = \hat{w}_0 + \sum_{i=1}^n x_i \hat{w}_i$$
$$\hat{y} = \mathbf{x}^T \hat{\mathbf{w}}$$

# Generalization

- ▶ The central challenge in machine learning is to design an algorithm which will perform well on new data (different from the training set data).
- ▶ This ability is called **generalization**.
- ▶ **Training error** is the error computed on the training set.
- ▶ During the training (learning) we aim at reducing the training error.
- ▶ If that is the end goal, we only have an optimization problem, not a machine learning one.



# Generalization error

- ▶ **Generalization error**, also called **test error** is defined as the expected error on new, previously unseen data.
- ▶ Unlike in simple optimization, in machine learning our main goal is to minimize the **generalization error**.
- ▶ Usually the generalization error is estimated by measuring the performance on a **test data set** which must be independent from the training set.



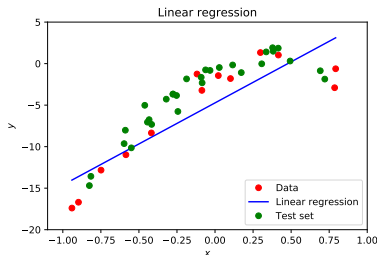
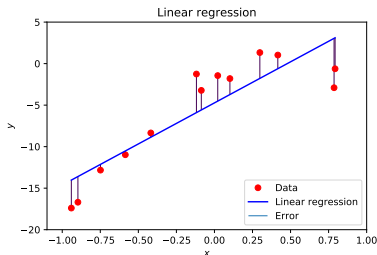
# Example: Linear regression

- Previously, we trained the model by minimizing the training error

$$\frac{1}{m(\text{train})} \left\| \mathbf{X}^{(\text{train})} \hat{\mathbf{w}} - \mathbf{y}^{(\text{train})} \right\|_2^2$$

- We would like actually to minimize the test error

$$\frac{1}{m(\text{test})} \left\| \mathbf{X}^{(\text{test})} \hat{\mathbf{w}} - \mathbf{y}^{(\text{test})} \right\|_2^2$$



# Statistical learning theory

- ▶ **Statistical learning theory** provides methods to mathematically reason about the performance on the test set although we can observe only the training set.
- ▶ This is possible under some assumptions about the data sets
  - ▶ The training and test data are generated by drawing from a probability distribution over data sets. We refer to that as **data-generating process**.
  - ▶ **i.i.d. assumptions**
    - ▶ Examples in each data sets are **independent** from each other.
    - ▶ The training data set and the test data set are **identically distributed**, i.e., drawn from the same probability distribution.

## Discussion point 3

Can you name a scenario in medical image analysis practice where the i.i.d. assumptions are bound to be broken?

# Underfitting and overfitting

- ▶ The factor that determines how well a machine algorithm will perform is its ability to
  1. Make the training error small.
  2. Make the difference between the training and test error small.
- ▶ These two factors correspond to the two central challenges in machine learning: **underfitting** and **overfitting**.
- ▶ Underfitting occurs when the model is not able to produce a sufficiently small training error.
- ▶ Overfitting occurs when the gap between the training and test errors is too large.

# Model capacity

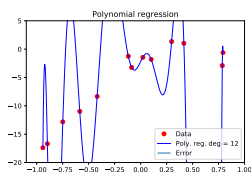
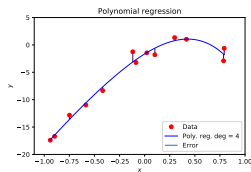
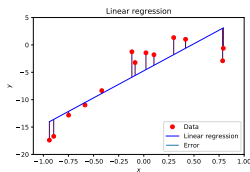
- ▶ A **capacity of the model** is its ability to fit a wide variety of functions.
- ▶ Low capacity models struggle to fit the training set (underfitting).
- ▶ Models with high capacity have danger to overfit the training data (e.g., by “memorizing” training samples).
- ▶ The capacity can be controlled by choosing its **hypothesis space**, i.e. the set of functions from which the learning algorithm is allowed to select the solution.
- ▶ Example: The linear regression algorithm has the set of all linear functions as its hypothesis space.

# Polynomial regression

- ▶ The linear regression algorithm can be generalized to include all polynomial functions instead of just the linear ones.
- ▶ The linear regression model is then just a special case restricted to a polynomial of degree one:  $\hat{y} = b + wx$ .
- ▶ Moving to degree two to we obtain:  $\hat{y} = b + w_1x + w_2x^2$ .
  - ▶ This can be seen as adding a new feature  $x^2$ .
  - ▶ In fact, we can generalize this approach to create all sorts of hypothesis spaces, e.g.:  $\hat{y} = b + w_1x + w_2 \sin(x) + w_3\sqrt{x}$ .
- ▶ The **output** is still a **linear** function of the parameters, so in principle it can be trained in the same way as the linear regression.

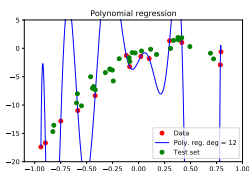
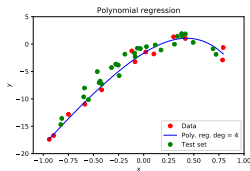
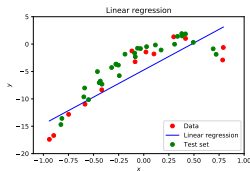
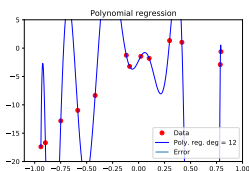
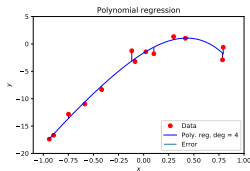
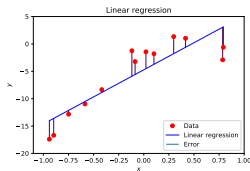
# Polynomial regression

A comparison of a linear, degree-4, and degree-12 polynomials as predictors



# Polynomial regression

A comparison of a linear, degree-4, and degree-12 polynomials as predictors





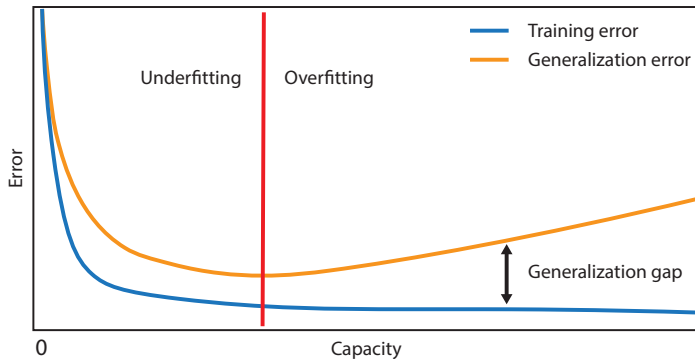
# Overfitting and underfitting in polynomial estimation

- ▶ Models with low capacity are not up to the task.
- ▶ Models with high-capacity can solve a complex task, but when the capacity is too high for the concrete (training) task there is the danger of overfitting.
- ▶ In our example: the linear function is unable to capture the curvature so it underfits.
- ▶ The degree-12 predictor is capable of fitting the training data, but it also able to find infinitely many functions that pass through the same points, so it has high probability of overfitting.
- ▶ The degree-4 function is the right solution and it generalizes well on the new data.

# Generalization and capacity

- ▶ Simpler functions generalize more easily, but we still need to choose a sufficiently complex hypothesis (function) to obtain small training error.
- ▶ Typically training error decreases with the increase of the model capacity until an (asymptotic) value is reached.
- ▶ The generalization error is U-shaped with the capacity range split in an underfitting and an overfitting zone(see next slide).

# Generalization and capacity



# Training set size

- ▶ Training and generalization error vary as the size of the training data set varies.
- ▶ Expected generalization error never increases as the size of the training set increases.
- ▶ Any fixed parametric model will asymptotically approach an error value that exceeds the so called Bayes error.
- ▶ It is possible for the model to have optimal capacity and still have a large gap between training and generalization errors.
- ▶ In that case the gap usually can be reduced with increasing the number of training examples.

# Training set size

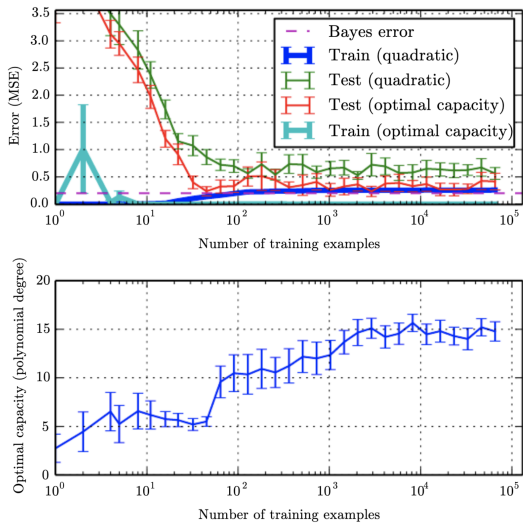


Figure from Goodfellow et al., *Deep Learning*

# The No Free Lunch theorem

- ▶ **No Free Lunch Theorem** for machine learning (Wolpert, 1996):  
Averaged over all possible data-generating distributions every classification algorithm has the same error rate when tested on new unobserved data.
- ▶ In some sense, no machine algorithm is universally better than any other algorithm.
- ▶ An interesting, but mainly theoretical result.
- ▶ In practice we often have an information about the probability distributions we deal with and can tailor our algorithms to perform well with particular distributions.

# Regularization

- ▶ In addition to increasing and decreasing of the hypothesis space, i.e., the capacity, we can influence the learning algorithm by **giving preference to one solution over another in the hypothesis space**.
- ▶ In case both functions are eligible we can define a condition to express preference about one of the functions.
- ▶ The unpreferred solution is chosen only if it gives significantly better performance with the training data.
- ▶ *More on regularization in the next lecture.*

# Model selection

Materials:

- ▶ Chapter 1.5.3 from Goodfellow et al., *Deep Learning*



# Hyperparameters and validation sets

- ▶ **Hyperparameters** are settings that can be used to control the behaviour of the algorithm.
- ▶ In general, the hyperparameters are not modified by the learning algorithm itself.
- ▶ **Example:** In **polynomial regression** the degree of the polynomial is a **capacity** hyperparameter.
- ▶ A setting can be chosen to be hyperparameter when it is **difficult to optimize** or - more often - when its derivation from the training set **can lead to overfitting**.
  - ▶ Example: in polynomial regression we can always fit the data better with a higher degree polynomial.

# Choice of training, validation, and test sets

- ▶ The **validation set** is used during training to predict the behaviour (generalization error) of the algorithm on new data, i.e., on the test set and to choose the hyperparameters.
- ▶ Ideally these two sets are disjoint.
- ▶ The validation set is chosen from the training data.
- ▶ The training data is split in two disjoint subsets.
- ▶ One subset is used to learn the parameters of the algorithm and the other is the validation set.
- ▶ The subset used to learn the parameters is still typically called a **training set**.

# Choice of training, validation, and test sets

- ▶ Since the validation set is used to determine the hyperparameters it will typically underestimate the generalization error.
- ▶ However, it will usually better predict the generalization error than the training set.
- ▶ After the completion of the hyperparameters optimization we can estimate the generalization error using the test data.
- ▶ In practice the testing should be done also on different test data to avoid the test data becoming “stale”.

# Choice of training, validation, and test sets

Training

Used to find the optimal **parameters** of the model.

$$w$$

Validation

Used to find the optimal **model** (hyper-parameters).

$$f(\cdot)$$

Test

Used to estimate the **performance** of the optimal model.

$$||\hat{y} - y||$$

## Discussion point 4

How large should the training, validation and testing datasets be as a percentage (%) of the total available data?

# Cross-validation

- ▶ Dividing the data set into disjoint training and test sets can result in a result in a too small validation and/or test set.
- ▶ In such cases all data is used to estimate the generalization error.
- ▶ We use procedures that repeat the training and testing on different randomly chosen subsets or splits of the original data set.
- ▶ The most common such procedure is the **k-fold cross-validation**.

# Cross-validation

- ▶ The original data is partitioned into  $k$  (disjoint) subsets.
- ▶ The average error can be estimated by taking the average over  $k$  trials.
- ▶ In trial  $i$ , the  $i$ -th subset is used as test set and the rest as training set.
- ▶ Problem: no unbiased estimators of the variance of such average error exist, but there are approximations that are used in practice.

# Expectation (recap)

- ▶ The **expectation** or **expected** value of a function  $f(x)$  with respect to a probability distribution  $P(x)$  is the average value of  $f$  over all values  $x$  assuming they are drawn from  $P$

$$\mathbb{E}_{x \sim P}[f(x)] = \sum_x P(x)f(x)$$

$$\mathbb{E}_{x \sim P}[f(x)] = \int p(x)f(x)dx$$



## Variance (recap)

- ▶ The **variance** gives a measure of variation of the values of a random variable  $x$

$$\text{Var}(f(x)) = \mathbb{E}[(f(x) - E[f(x)])^2]$$

# Bias and variance trade-off

Materials:

- ▶ Chapter 1.5.4 from Goodfellow et al., *Deep Learning*

# Point estimation

- ▶ For efficient design of learning algorithms it is useful to have formal characterizations of notions like generalization, overfitting and underfitting.
- ▶ To this end we introduce some definitions.
- ▶ **Point estimation** is the attempt to provide the single "best" prediction of some quantity of interest.
- ▶ The quantity of interest can be a single parameter, parameter vector of some model, e.g., the weights  $\mathbf{w}$  in the linear regression model.
- ▶ It can also be a whole function, e.g., the linear function or polynomial of some degree, like in the polynomial regression.

# Point estimation

- ▶ Given a parameter  $\theta$  we denote its point estimate with  $\hat{\theta}$ .
- ▶ As usual, let  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  be  $m$  independent and identically distributed (i.i.d.) data points.
- ▶ A **point estimator** or **statistic** is any function of the data

$$\hat{\theta}_m = g(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)})$$

- ▶ This definition is very general. For instance, that the value returned by  $g$  need not be close to the true value  $\theta$ . Also  $g$  might return a value which is outside the values that  $\theta$  is allowed to have.

# Point estimation

- ▶ Of course, a good estimator is still a function that returns values close to  $\theta$ .
- ▶ Since the data is drawn from a random process, point estimate  $\hat{\theta}$  is considered to be a random variable and  $\theta$  is fixed, but unknown parameter.

# Function estimation

- ▶ In **function estimation**, we assume that there is a (true) function that describes the (approximate) relationship between  $\mathbf{x}$  and  $\mathbf{y}$

$$\mathbf{y} = f(\mathbf{x}) + \epsilon$$

where  $\epsilon$  is the part of  $\mathbf{y}$  which is not predictable from  $\mathbf{x}$

- ▶ The goal is to find the **function estimate (model)**  $\hat{f}$  which is a good approximation of  $f$ .
- ▶ The linear regression and polynomial regression can be seen both illustrate scenarios that can be interpreted as either estimating a parameter  $\mathbf{w}$  or estimating a function  $\hat{f}$ .

- ▶ A bias of an estimator  $\hat{\theta}_m$  is defined as

$$\text{bias}(\hat{\theta}_m) = \mathbb{E}(\hat{\theta}_m) - \theta$$

where the expectation is over the data and  $\theta$  is the true underlying value.

- ▶ An estimator  $\hat{\theta}_m$  is **unbiased** if  $\text{bias}(\hat{\theta}_m) = 0$ . Note that this implies  $\mathbb{E}(\hat{\theta}_m) = \theta$ .
- ▶  $\hat{\theta}_m$  is **asymptotically unbiased** if  $\lim_{m \rightarrow \infty} \text{bias}(\hat{\theta}_m) = 0$  (implying  $\lim_{m \rightarrow \infty} \mathbb{E}(\hat{\theta}_m) = \theta$ ).

# Bias: example

- **Example:** Consider samples  $\{x^{(1)}, \dots, x^{(m)}\}$  i.i.d distributed according to the Gaussian distribution

$$p(x^{(i)}; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{x^{(i)} - \mu}{\sigma^2}\right)$$

- The **sample mean** is a common estimator of the Gaussian mean parameter

$$\hat{\mu}_m = \frac{1}{m} \sum_{i=1}^m x(i)$$



## Bias: example

We compute the bias as expectation by substituting the Gaussian distribution in the formula

$$\begin{aligned}\text{bias}(\mu_m) &= \mathbb{E}[\mu_m] - \mu \\ &= \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m x^{(i)}\right] - \mu \\ &= \left(\frac{1}{m} \sum_{i=1}^m \mathbb{E}[x^{(i)}]\right) - \mu \\ &= \left(\frac{1}{m} \sum_{i=1}^m \mu\right) - \mu \\ &= \mu - \mu = 0\end{aligned}$$

The sample mean is an unbiased estimator of Gaussian mean parameter.

## Bias: example

- ▶ **Example:** Estimators of the variance of a Gaussian distribution
- ▶ We compare two different estimators of the variance  $\sigma^2$  parameter
- ▶ **Sample variance**

$$\hat{\sigma}^2 = \frac{1}{m} \sum_1^m \left( x^{(i)} - \hat{\mu}_m \right)^2$$

where  $\hat{\mu}$  is the sample mean.

- ▶ We are interested in computing

$$\text{bias}(\hat{\sigma}_m^2) = \mathbb{E}[\hat{\sigma}_m^2] - \sigma^2$$

## Bias: example

- ▶ First we evaluate  $\mathbb{E}[\hat{\sigma}_m^2]$ :

$$\mathbb{E}[\hat{\sigma}_m^2] = \mathbb{E} \left[ \frac{1}{m} \sum_1^m \left( x^{(i)} - \hat{\mu}_m \right)^2 \right] = \frac{m-1}{m} \sigma^2$$

- ▶ Back to the bias

$$\text{bias}(\hat{\sigma}_m^2) = \mathbb{E}[\hat{\sigma}_m^2] - \sigma^2 = \frac{m-1}{m} \sigma^2 - \sigma^2 = -\frac{\sigma^2}{m}$$

- ▶ Therefore the sample variance is a **biased** estimator.

## Bias: example

- ▶ The **unbiased variance estimator** is defined as

$$\tilde{\sigma}^2 = \frac{1}{m-1} \sum_1^m \left( x^{(i)} - \hat{\mu}_m \right)^2$$

- ▶ Indeed

$$\mathbb{E}[\tilde{\sigma}_m^2] = \mathbb{E} \left[ \frac{1}{m-1} \sum_1^m \left( x^{(i)} - \hat{\mu}_m \right)^2 \right] = \frac{m-1}{m-1} \sigma^2 = \sigma^2$$

and the bias is 0.

# Variance and standard error

- ▶ Another important feature of an estimator is its variance.
- ▶ The **variance** of an estimator is simply its statistical variance  $\text{Var}(\hat{\theta})$  over the training set as a random variable.
- ▶ Alternatively we can compute the **standard error** (the square root of the variance)  $\text{SE}(\hat{\theta})$ .
- ▶ The variance or the standard error provide a measure how much the estimate would vary as we resample the data independently from the underlying data generating process.
- ▶ We would prefer a relatively low variance of the estimator.

# Variance and standard error

- ▶ The standard error of the mean estimator is given as

$$\text{SE}(\hat{\mu}) = \sqrt{\text{Var} \left[ \frac{1}{m} \sum_{i=1}^m x^{(i)} \right]} = \frac{\sigma}{\sqrt{m}}$$

where  $\sigma$  is the true variance of the distribution, i.e., the samples  $x^{(i)}$ .

- ▶ Neither the square root of the sample variance nor the square root of the unbiased estimator of the variance give an unbiased estimate of the standard deviation.
- ▶ Both approaches underestimate the true standard deviation.
- ▶ However, for large  $m$  the approximation works quite well.

# Variance and standard error

- ▶ In machine learning the generalization error is estimated based on the sample mean of the error on the test set.
- ▶ The accuracy of the estimate depends on the number of the examples.
- ▶ From the statistical theory (central limit theorem) we know that the mean is distributed with normal distribution for which we can establish confidence intervals.
- ▶ For instance, the 95% confidence interval is given by

$$[\hat{\mu}_m - 1.96SE(\hat{\mu}_m), \hat{\mu}_m + 1.96SE(\hat{\mu}_m)]$$

- ▶ Then we can say that algorithm A is better than algorithm B if the confidence upper bound for the error of A is less than the corresponding lower bound of B.

# Trading off bias and variance to minimize mean squared error

- ▶ Bias and variance measure two different sources of error in an estimator.
- ▶ Bias measures the expected deviation with the true value of the estimator.
- ▶ Variance provides a measure of the deviation from the expected value of the estimator depending on the particular data sampling.



# Trading off bias and variance to minimize mean squared error

- ▶ Often we need to make a trade-off between these two.
- ▶ The most common way to do this is via cross-validation.
- ▶ An alternative is to compare the **mean squared error** (MSE) of the estimates.

$$\text{MSE} = \mathbb{E}[(\hat{\theta}_m - \theta)^2] = \text{bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m)$$

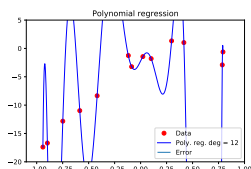
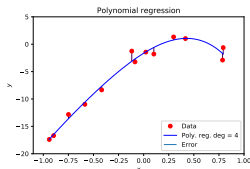
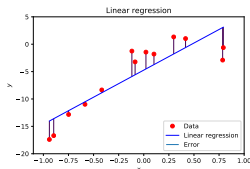
- ▶ The smaller MSE the better - so minimizing both the bias and variance is always preferable.

# Bias and variance

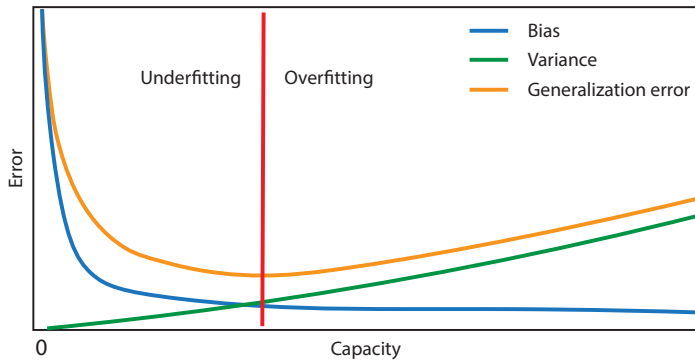
- ▶ Our original goal was to provide a mathematical support for the notions of capacity, underfitting, and overfitting.
- ▶ Indeed there is a close relationship between these three concepts and bias and variance.
- ▶ When generalization error is measured by MSE (and hence indirectly via bias and variance) increasing capacity tends to increase variance and decrease bias.
- ▶ Again the generalization as a function of capacity is given by an U-shaped curve.

# Discussion

How will the estimated regression model change when one training data point is replaced with another one?



# Bias and variance



# Consistency

- ▶ So far we considered fixed size of the training data sets.
- ▶ We expect that as the number  $m$  of training examples grows the estimators will converge to the true value of the parameters.
- ▶ More formally this is captured in the notion of **consistency**

$$\text{plim}_{m \rightarrow \infty} \hat{\theta}_m = \theta$$

where plim denotes convergence in probability: for any  $\epsilon > 0$ ,  $P(|\hat{\theta}_m - \theta| > \epsilon) \rightarrow 0$  as  $m \rightarrow \infty$ .

- ▶ For consistent models the bias decreases as  $m$  increases, however a decreasing bias (when  $m$  increases) does not imply consistency.

# Maximum likelihood estimation

Materials:

- ▶ Chapter 1.5.5 from Goodfellow et al., *Deep Learning*

# Maximum likelihood estimation

- ▶ We would like to have some principle from which we can derive good estimator functions for a large scale of models.
- ▶ The **maximum likelihood estimation** is the most common such principle.
- ▶ Given observation data and a corresponding (statistical) model our goal is to find the parameter vector which imply the highest probability to obtain the data.

# Maximum likelihood estimation

- ▶ Consider a set of  $m$  examples  $\mathbb{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}\}$  drawn independently from the true but unknown distribution  $p_{\text{data}}(\mathbf{x})$ .
- ▶ Let  $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$  be a parametric family of probability distributions, i.e., for each  $\boldsymbol{\theta}$  we get a different distribution  $p_{\text{model}}$ .
- ▶  $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$  maps any configuration  $\mathbf{x}$  to a real number estimating the (true) probability  $p_{\text{data}}(\mathbf{x})$



# Maximum likelihood estimation

- ▶ The maximum likelihood estimator for  $\theta$  is then defined as

$$\theta_{\text{ML}} = \operatorname{argmax}_{\theta} p_{\text{model}}(\mathbb{X}; \theta) = \operatorname{argmax}_{\theta} \prod_{i=1}^m p_{\text{model}}(\mathbf{x}^{(i)}; \theta)$$

Note that also the empirical distribution  $\hat{p}_{\text{data}}$  is implicitly present in the formula through  $\mathbf{x}^{(i)}$ .

- ▶ A more convenient equivalent optimization problem is obtained by taking logarithm of the product

$$\theta_{\text{ML}} = \operatorname{argmax}_{\theta} p_{\text{model}}(\mathbb{X}; \theta) = \sum_{i=1}^m \log p_{\text{model}}(\mathbf{x}^{(i)}; \theta)$$

# Maximum likelihood estimation

- ▶ We can further rescale by dividing the expression by  $m$

$$\theta_{\text{ML}} = \operatorname{argmax}_{\theta} p_{\text{model}}(\mathbb{X}; \theta)$$

$$\theta_{\text{ML}} = \operatorname{argmax}_{\theta} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{x}; \theta)$$

- ▶ In this way the problem is expressed as an equivalent expectation problem (now the empirical distribution  $\hat{p}_{\text{data}}$  becomes explicit).

# Maximum likelihood estimation

- ▶ Perhaps more straightforwardly, the maximum likelihood estimation can be seen as minimizing the dissimilarity between  $\hat{p}_{\text{data}}$  and  $p_{\text{model}}$ .
- ▶ The degree of dissimilarity is given by the KL-divergence

$$D_{\text{KL}}(\hat{p}_{\text{data}} \| p_{\text{model}}) = \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [\log \hat{p}_{\text{data}}(\mathbf{x}) - \log p_{\text{model}}(\mathbf{x})]$$

- ▶ Only the term of the right is function of the model, so it is the only one which needs to be minimized

$$-\mathbb{E}_{\mathbf{x} \sim \hat{p}_{\text{data}}} [\log p_{\text{model}}(\mathbf{x})]$$

which is equivalent with the maximization problem from the previous slide.

- ▶ It boils down to minimizing the **cross-entropy** between the two distributions.

# Maximum likelihood estimation

- ▶ **The maximum likelihood estimation can be seen as an attempt to make the model distribution  $p_{\text{model}}$  to match the empirical distribution  $\hat{p}_{\text{data}}$ .**
- ▶ Ideally we would like to match the data generating distribution  $p_{\text{data}}$ , but we do not have access to it.

# Conditional log likelihood and mean square error

- ▶ The maximal likelihood estimator can be generalized to estimate a conditional probability  $P(\mathbf{y} \mid \mathbf{x}; \boldsymbol{\theta})$ .
- ▶ Let all inputs be given by  $\mathbf{X}$  and all observed outputs by  $\mathbf{Y}$ . Then the conditional maximum likelihood estimator is

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\boldsymbol{\theta}} P(\mathbf{Y} \mid \mathbf{X}; \boldsymbol{\theta})$$

- ▶ If the examples are assumed to be i.i.d., then this can be decomposed into

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\boldsymbol{\theta}} \sum_{i=1}^m \log P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}; \boldsymbol{\theta})$$

## Example: linear regression as maximum likelihood

- ▶ The linear regression seen as an algorithm that learns to take an input  $\mathbf{x}$  and produce output  $\hat{y}$ .
- ▶ This function from  $\mathbf{x}$  to  $\hat{y}$  is chosen to minimize the mean squared error.
- ▶ This criterion was introduced more or less arbitrarily.
- ▶ We revisit linear regression from the point of view of maximal likelihood.
- ▶ We think of the model as producing a conditional distribution  $p(y \mid \mathbf{x})$  instead of a single prediction  $\hat{y}$ .

## Example: linear regression as maximum likelihood

- ▶ With an infinitely large training set we might see several examples with the same input  $\mathbf{x}$  but different  $y$ .
- ▶ The learning algorithm needs to fit the distribution to all these  $y$  corresponding to the same  $\mathbf{x}$ .
- ▶ To derive the linear regression algorithm we assume  $p(y | \mathbf{x}) = \mathcal{N}(y; \hat{y}(\mathbf{x}; \mathbf{w}), \sigma^2)$ , where  $\hat{y}(\mathbf{x}; \mathbf{w})$  gives the (prediction of the) mean of the normal distribution and  $\sigma$  is fixed to some chosen constant.
- ▶ The parameter vector  $\theta$  corresponds in this case to  $\mathbf{w}$ .

## Example: linear regression as maximum likelihood

- ▶ By substituting (the full Gaussian function version of)  $p(y \mid \mathbf{x})$  in the conditional log-likelihood formula we obtain

$$\sum_{i=1}^m \log p(\mathbf{x}^{(i)} \mid y^{(i)}; \boldsymbol{\theta}) = -m \log \sigma - \frac{m}{2} \log(2\pi) - \sum_{i=1}^m \frac{\|\hat{y}^{(i)} - y^{(i)}\|^2}{2\sigma^2}$$

where  $\hat{y}^{(i)}$  is the linear regression on the  $i$ -th input  $\mathbf{x}^{(i)}$ .

- ▶ By comparing with the mean squared error

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^m \|\hat{y}^{(i)} - y^{(i)}\|^2$$

one can see that maximizing the log-likelihood with respect to  $\mathbf{w}$  results with the same estimate of  $\mathbf{x}$  as minimizing MSE.

(The third term in the log-likelihood formula, needs to be as small as possible.)



# Properties of maximum likelihood

- ▶ It can be shown that the maximum likelihood estimator is the best asymptotically, i.e. as  $m \rightarrow \infty$ , in terms of its convergence rate.
- ▶ **Property of consistency:** as the number of training examples approaches infinity the maximum likelihood estimate of a parameter converges towards the true parameter value.
- ▶ The maximum likelihood estimator has the property of consistency provided:
  - ▶ The true distribution  $p_{\text{data}}$  is in the model family  $p_{\text{model}}(\cdot; \theta)$
  - ▶  $p_{\text{data}}$  corresponds to exactly one value of  $\theta$

# Model evaluation

Materials:

- ▶ Fawcett, “An introduction to ROC analysis”

# Model evaluation

- ▶ To quantitatively evaluate a machine learning algorithm we need to define a **performance measure**.
- ▶ Usually the performance measure is specific to the task carried out by the algorithm.
- ▶ For classification tasks a natural measure is the model **accuracy**.
- ▶ The **accuracy** is defined as the proportion of examples for which the model produces the correct output.
- ▶ An equivalent (complementary) measure is the **error rate** defined as the proportion of incorrect outputs.

# Model evaluation

- ▶ The best way to evaluate a machine learning algorithm is by applying it to a **test set** data which has not been seen before.
- ▶ Ideally there should be **no overlap** between the **test set** and the **training set** used to obtain the model.

# Binary classification

- ▶ We consider **binary classification** problems, i.e., problems using only two classes/
- ▶ Formally each input example  $x^{(i)}$  needs to be mapped into one element of the set  $\{\mathbf{p}, \mathbf{n}\}$  of **true** classes.
- ▶ A **classification model (classifier)** is a function from the input examples to the set  $\{\mathbf{Y}, \mathbf{N}\}$  of **predicted classes** or **hypothesized classes**.
- ▶  $\mathbf{p}, \mathbf{n}$  correspond to  $\mathbf{Y}, \mathbf{N}$ , respectively.

# Binary classification

- ▶ For a given classifier there are four possible outcomes.
- ▶ If the true class of  $x^{(i)}$  is  $\mathbf{p}$  and the predicted class is  $\mathbf{Y}$  then we have a **true positive** (TP); if it was classified  $\mathbf{N}$ , then we have a **false negative** (FN).
- ▶ Symmetrically, a  $x^{(i)}$  with true class  $\mathbf{n}$  which is assigned a predicted class  $\mathbf{N}$  is a **true negative** (TN); if the predicted class is  $\mathbf{Y}$ , then it is a false positive (FP).

# Confusion matrix

These four combinations can be put together in a **confusion matrix**, also called **contingency table**.

		True class	
		<i><b>p</b></i>	<i><b>n</b></i>
Predicted class	<i><b>Y</b></i>	True positives (TP)	False positives (FP)
	<i><b>N</b></i>	False negatives (FN)	True negatives (TN)

# Binary classifications metrics

- ▶ Using the four basic categories of prediction outcomes (TP, FP, TN, FN) we can derive various **measures** of performance of classification models.
- ▶ For instance the accuracy can be defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

- ▶ Also quite frequently used measures are

$$\text{Sensitivity} = \frac{TP}{TP + FN} \quad \text{Specificity} = \frac{TN}{TN + FP} \quad \text{Precision} = \frac{TP}{TP + FP}$$



# Binary classification metrics

- ▶ Sensitivity is also called **recall**, **true positive rate** or **hit rate**.
- ▶ In medical contexts the sensitivity can be interpreted as a measure of the extent to which diseased individuals are correctly diagnosed.
- ▶ In general: measures the proportion of the target group the method is able to detect, i.e. how sensitive is to this group.
- ▶ Specificity is also called **true negative rate** or **selectivity**.
- ▶ In medical contexts the specificity can be interpreted as a measure of the extent to which healthy individuals are correctly diagnosed.
- ▶ The precision tells us which proportion of the positive predictions is correct.

## Discussion point 6

You have developed a method for some image analysis diagnostic task that has very high sensitivity (e.g. 0.99) but relatively low specificity (e.g. 0.25).

Can this be still a useful tool for clinicians and if so in what context?

# Binary classification metrics

- ▶ Sometimes the above mentioned measures are not sufficient.
- ▶ For example, in a population in which the percentage of healthy individuals is much larger than the diseased individuals, it is easy to achieve high specificity by trivially classifying each patient as healthy.
- ▶ We can obtain more objective evaluation by combining metrics.
- ▶ The metrics  $F_1$  is the harmonic mean (average) of the precision and recall (sensitivity)

$$\frac{2}{F_1} = \frac{1}{Precision} + \frac{1}{Recall} \text{ or } F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

# Areas under the curve measures

- ▶ (Binary) classifications often depend on some parameter (e.g., threshold).
- ▶ Hence one way to combine two metrics is by assigning them to the axes of a coordinate system and varying this parameter to construct a graphical plot.
- ▶ We obtain a curve (actually, most of the time series of points) such that each point corresponds to a particular parameter value.
- ▶ The area under the curve is a measure of how good is the classification.

# Receiver Operating Characteristic (ROC) curve

- ▶ The ROC curve plots the true positive rate (sensitivity) versus the false positive rate (1 - specificity).

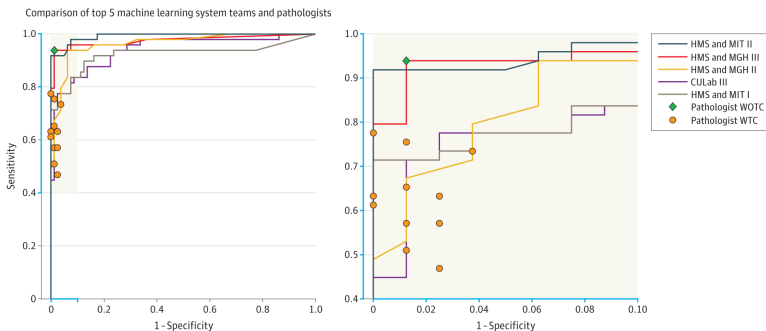


Figure from Bejnordi et al., “Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer”

# Receiver Operating Characteristic (ROC) curve

- ▶ There are several characteristic points in the ROC space:
  - ▶  $(0,0)$  corresponds to the strategy of never making a positive classification.
  - ▶  $(1,1)$  is the opposite: unconditionally issuing a positive classification.
  - ▶  $(0,1)$  represents perfect classification.
  - ▶ Obviously we strive to achieve this ideal point as a result have as much as possible area under the curve covered (ideally it should cover the whole square corresponding to the ROC space)
- ▶ A less common example of a measure combination into a graphical plot is the precision-recall plot (recall on the x-axis, precision on the y-axis).

# Supervised and unsupervised learning algorithms

Materials:

- ▶ Chapters 1.5.6 and 1.5.7 from Goodfellow et al., *Deep Learning*

# Supervised learning algorithms

- ▶ Learning algorithms that learn based on a given training examples  $\mathbf{x}$  and their corresponding outputs  $\mathbf{y}$ .
  - ▶ Linear and logistic regressions
  - ▶ Support vector machines
  - ▶  $k$ -nearest neighbours
  - ▶ Decision trees



# Unsupervised learning algorithms

- ▶ Unsupervised algorithms experience only "features", but not supervision feedback.
- ▶ The distinction with the supervised algorithms is not always clear since there is no good test to distinguish if something is a feature or a target provided by the supervisor.
- ▶ Rule of thumb: in unsupervised algorithms no human annotation is needed for the training examples.
  - ▶ Principal component analysis
  - ▶  $k$ -means clustering
  - ▶ t-Distributed Stochastic Neighbor Embedding
  - ▶ Generative adversarial networks

# Ensambling

Materials:

- ▶ Chapter II.7.11 from Goodfellow et al., *Deep Learning*

# Bagging and other ensemble methods

- ▶ **Bagging** (short for **bootstrap aggregating**) is a technique for reducing of the generalization error by combining several models.
- ▶ Train models separately and let them vote on the right output.
- ▶ An example of a general strategy in machine learning called **model averaging**.
- ▶ Methods using this strategy are called *ensemble methods*.
- ▶ The rationale behind the combining of models is that usually different models will not make the same error.

# Bagging

- ▶ Different ensemble methods compose the ensemble of models in different ways.
- ▶ One way would be to choose the models, training algorithms and objective functions as different as possible.
- ▶ In contrast, bagging allows the same kind of model, algorithm and objective function to be reused several times.

# Bagging

- ▶ Bagging constructs  $k$  different data sets.
- ▶ Each data set
  - ▶ has the same size as the original set and
  - ▶ is constructed by sampling with repetition from the original data set
- ▶ For each data set a different model is produced.
- ▶ Each model reflects the differences between the (training) data sets.

# Bagging example

Training an “8 detector” with two resampled datasets: a “cartoon” example.

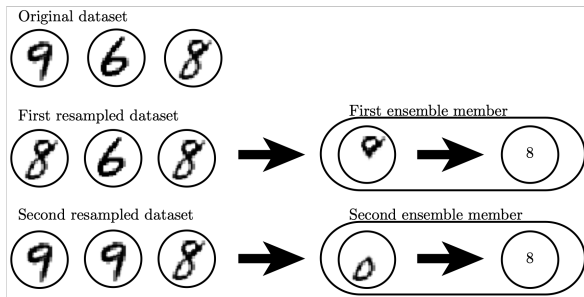


Figure from Goodfellow et al., *Deep Learning*

# Model averaging for neural networks

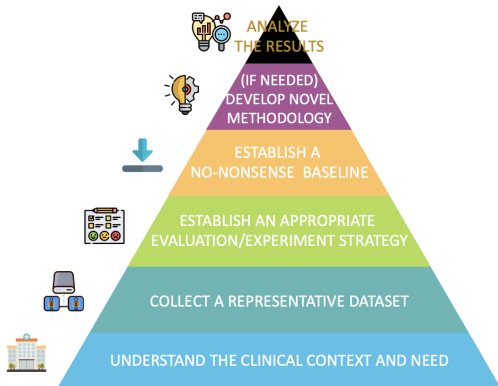
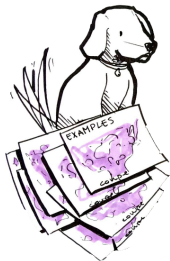
- ▶ Neural networks profit from model averaging even when they are trained on the same data set.
- ▶ This is because with random initialization, minibatches (subsets of the training set), hyperparameters, non-determinism in the implementation a sufficient variety between the models can be achieved.

# Model averaging in general

- ▶ In general, it is considered that model averaging always improves the generalization error.
- ▶ In theory, with sufficient computer memory and time one can always improve the results by combining several methods.
- ▶ Therefore, when testing/benchmarking (new) methods it is considered "fair" to use only a single model.
- ▶ Machine learning contests are usually won by using model averaging.
- ▶ **Boosting** is similar to ensembling, only the models (neural networks) are added **incrementally** to the ensemble.



# In summary...



# Important: introductory topics covered in video lecture

These topics are covered in the extended slide stack and video lecture (links in Cavas) and are exam material:






- ▶ Linear algebra
- ▶ Probability theory
- ▶ Maximum-likelihood estimation
- ▶ Supervised and unsupervised algorithms
- ▶ Ensambling

# Acknowledgements

The slides for this lecture were prepared by Mitko Veta and Dragan Bošnjacki.

Some of the slides are based on the accompanying lectures of Goodfellow et al., *Deep Learning*.

# References

-  Bejnordi, B. E. et al. “Diagnostic assessment of deep learning algorithms for detection of lymph node metastases in women with breast cancer”. In: *Jama* 318.22 (2017), pp. 2199–2210.
-  Fawcett, T. “An introduction to ROC analysis”. In: *Pattern recognition letters* 27.8 (2006), pp. 861–874.
-  Friedman, J., T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*. Springer series in statistics New York, 2001.
-  Goodfellow, I., Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
-  Kolter, Z. and C. Do. “Linear Algebra Review and Reference”. In: (2015).