Introduction to Machine Learning and Deep Learning with Python

Day 1

CNRS Formation Entreprise

19 – 21 June, 2024, Paris

Tabea Rebafka

Agenda

- Introduction to these 3 days of training
- 1st day: Supervised binary classification
- 2nd day: Decision trees and ensemble learning
- 3rd day: Deep learning

Outline of day 1

Introduction to this course

Binary classification

Logistic regression

Training of the classifier

Performance evaluation

Overfitting, penalization and cross validation

Neural networks: basic principles and architecture

Outline of day 1

Introduction to this course

Binary classification

Logistic regression

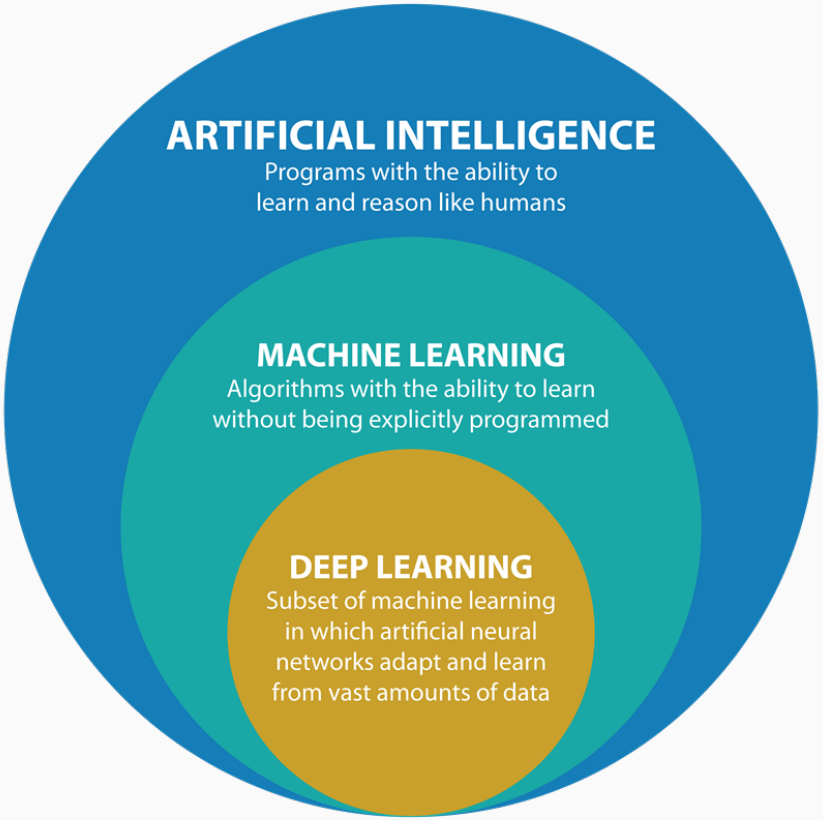
Training of the classifier

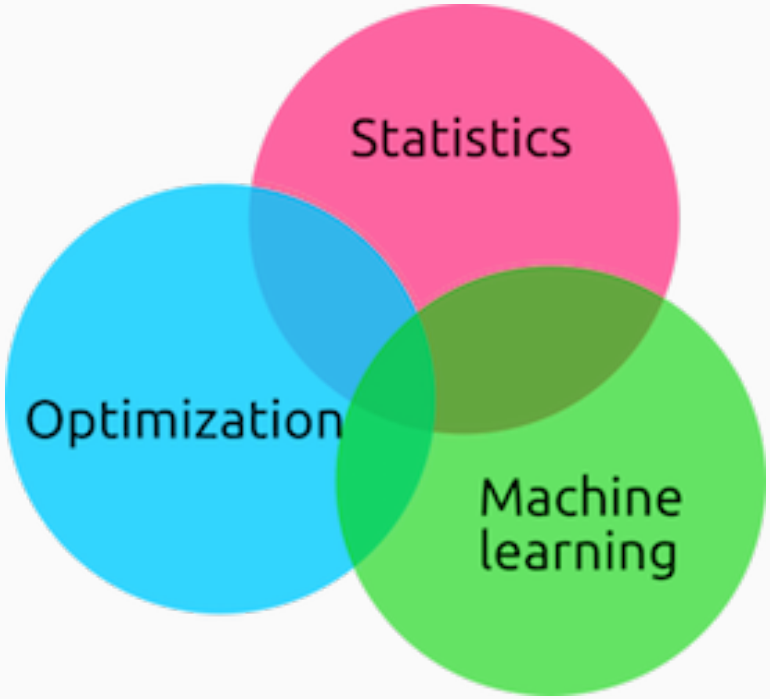
Performance evaluation

Overfitting, penalization and cross validation

Neural networks: basic principles and architecture

General picture



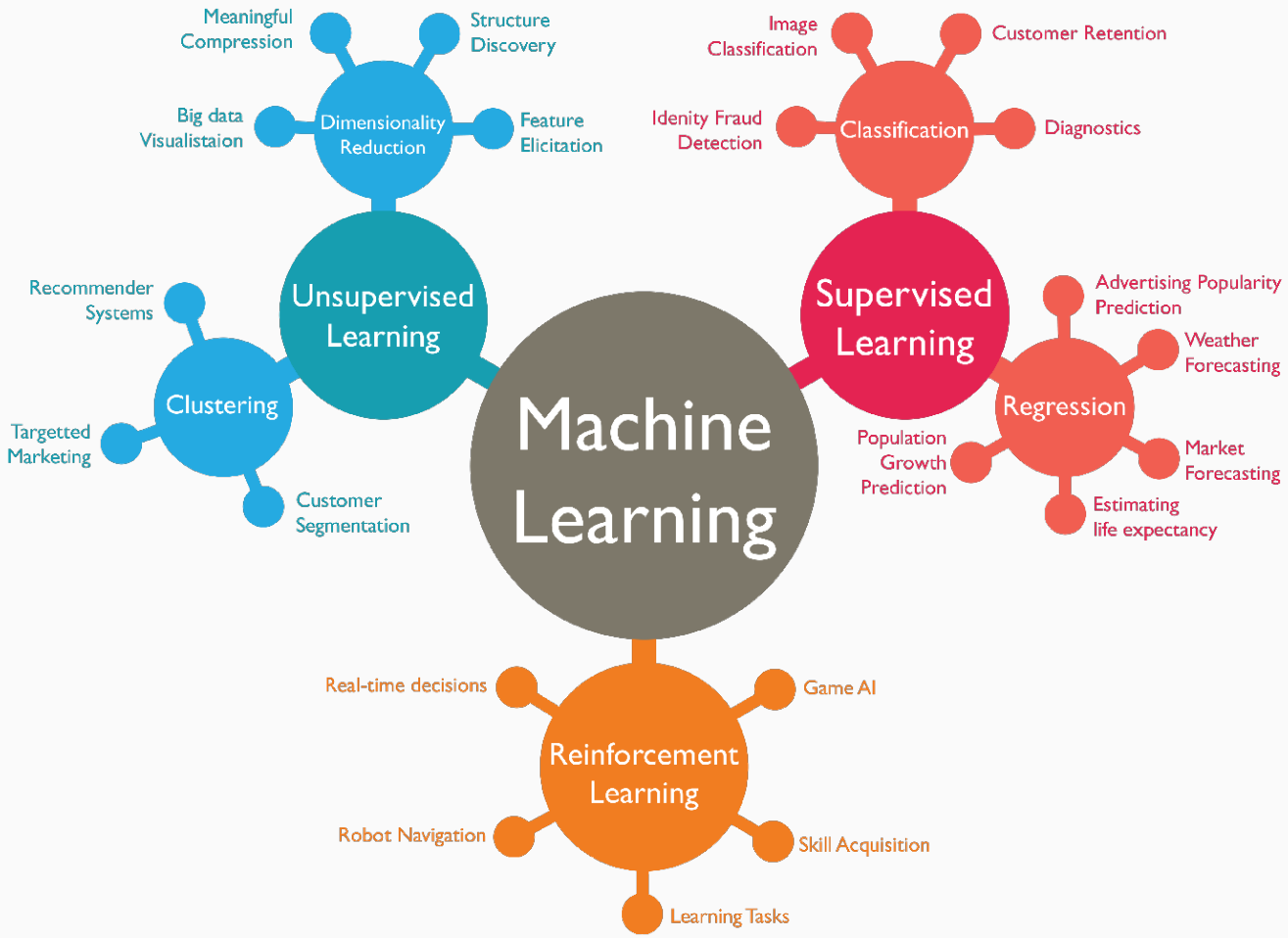


Machine Learning and Deep Learning: when and what for?

Data analysis for big data

- A lot of **promises** ... but it's **not magic**!
- First get to know the basic characteristics of your data: **descriptive statistics** remains a prerequisite!
- What we're going to do:
  - understand the fundamentals of **some** Machine Learning (ML) and Deep Learning (DL) techniques
  - see how it works on datasets
  - understand the limitations of these methods
- What we're **not** going to do:
  - list and test **all** possible ML and DL algorithms
  - describe all existing functions to perform an algorithm

Machine learning



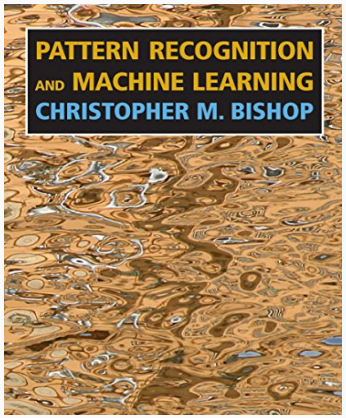
Python environment

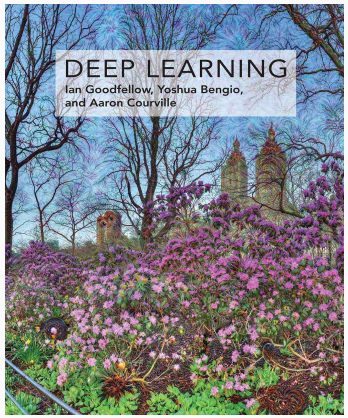
API	NumPy	Pandas	Scikit-Learn	PyTorch	Tensorflow	Keras[1]
Usage	math	Stats	ML and stats	ML & DL	DL	DL
Who?	all		academic world		Google	Google
API level		High		High	Low	High

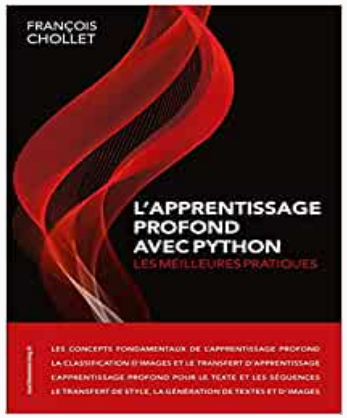
- High-level APIs are fairly abstracted, meaning that they are more generic and therefore limited in functionality. High-level API are easier to use.
- Low-level APIs are much more detailed and specific due to a low level of abstraction. Low-level APIs allow for finer control over application functions.

Some pointers i

Books







APIs websites

- Scikit-learn `https://scikit-learn.org/`
- Pandas `https://pandas.pydata.org/`
- PyTorch `https://pytorch.org`
- Keras `https://keras.io/`, Tensorflow `https://www.tensorflow.org/`

Some pointers ii

Others

- https://paperswithcode.com/
- https://stackoverflow.com/

Outline of day 1

Binary classification i

Classification

- **Supervised learning** with a binary label
- We observe data containing historical information on n **individuals**
- For each individual $i = 1, \ldots, n$, we observe
  - a **feature vector** $x_i \in \mathbb{R}^d$
  - a binary **label** y_i with possible values $y_i = 1$ or $y_i = -1$
- Data $\mathcal{D} = \{(x_i, y_i), i = 1, \ldots, n\}$ are considered to be i.i.d realizations of some random (X, Y)

Aim

- Given a new feature vector x_{new}, predict its unobserved label $y_{\text{new}} \in \{-1, 1\}$

Binary classification ii

Approach

- Use data $\{(x_i, y_i), i = 1, \ldots, n\}$ to learn a **classifier** $\hat{y} \in \{-1, 1\}$ that estimates label y_{new} of the new individual with features x_{new}

Classifier

- Function defined on the feature space $\mathbb{R}^d$, taking values in $\{-1, 1\}$:

$$\hat{y} : \mathbb{R}^d \to \{-1, 1\}$$

- $\hat{y}$ is learned on the data, that is, its form (or parameters) are **trained** on the historical data $\{(x_i, y_i), i = 1, \ldots, n\}$ containing observed labels
- For any new feature vector x_{new}, we can **make a prediction** of its label by evaluating the classifier $\hat{y}(x_{\text{new}})$

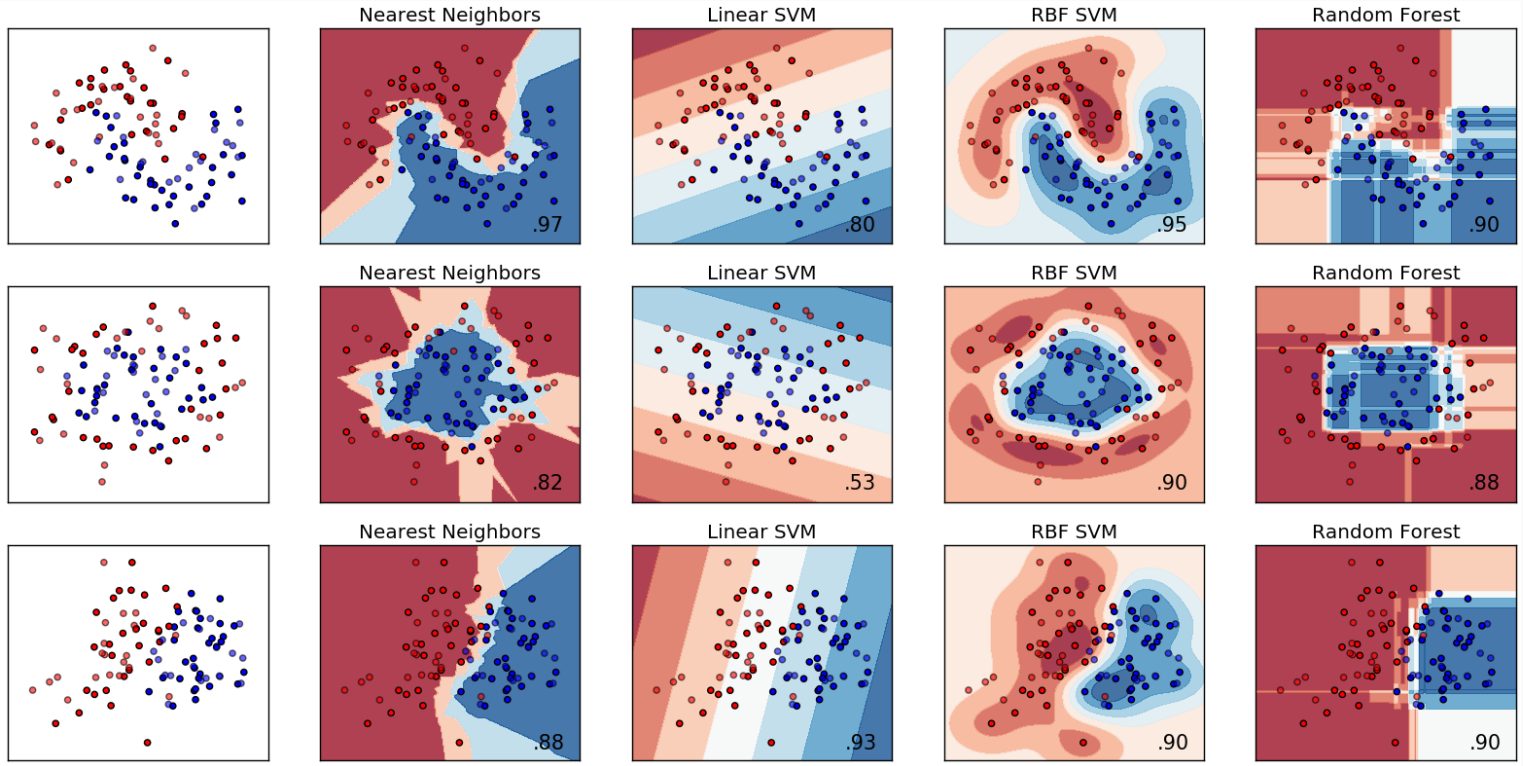
Outline of day 1

Model choices for classification i

Construction of a classifier

- Make assumptions on the relation between the features x and label y (**modelling**)
- Many possible model choices yielding very different classifiers

Model choices for classification ii



Model choices for classification iii

Statistical approach

- Model the distribution $\mathbb{P}(y|x)$ of y *knowing* x or *conditionally* on x (denoted $y|x$)
- Construct an estimator $\widehat{P}(y = 1|x)$ of the **probability** $\mathbb{P}(y = 1|x)$ for all $x \in \mathbb{R}^d$
  - this directly yields an estimator of $\mathbb{P}(y = -1|x)$ by $\widehat{P}(y = -1|x) := 1 - \widehat{P}(y = 1|x)$
- A **classifier** is defined by

$$\hat{y}(x_{\text{new}}) = \begin{cases} 1 & \text{if } \widehat{P}(y = 1|x_{\text{new}}) \geq t \\ -1 & \text{otherwise} \end{cases}$$

  for some **threshold** $t \in (0, 1)$
- The default choice $t = 1/2$ is used when the labels play a symmetric role

Model choices for classification iv

Notation

- For vectors $a, b \in \mathbb{R}^d$ (of same dimension d), the **inner product** between a and b is defined by

$$a^\top b = \langle a, b \rangle = \sum_{j=1}^{d} a_j b_j$$

- The 'simplest' (non-constant) function $\mathbb{R}^d \to \mathbb{R}$ is a **linear function**

$$x \mapsto x^\top w + b$$

  where $w \in \mathbb{R}^d$ is a vector of **weights** and $b \in \mathbb{R}$ is the **intercept**
- For $d = 1$ it's simply a straight line

Model choices for classification v

Goal

- Model the conditional probability $\mathbb{P}(y = 1|x)$

Generalized linear models

- Combine a **linear regression** model and a nonlinear **link function** g:

$$\mathbb{P}(y = 1|x) = g(x^\top w + b)$$

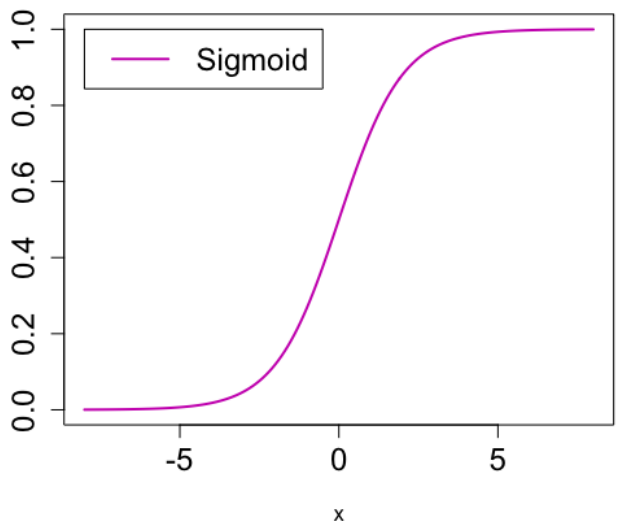
- The weights w and the intercept b are the **unknown model parameters** and have to be learned/estimated from the data

Model choices for classification vi

Logistic regression model or logit model

- By far the most widely used classification algorithm
- Use a **sigmoid function** (or **logistic function**) as link function:

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



- Thus, the **logistic regression model** (or **logit model**) is defined as

$$\mathbb{P}(y = 1|x) = \sigma(x^\top w + b)$$

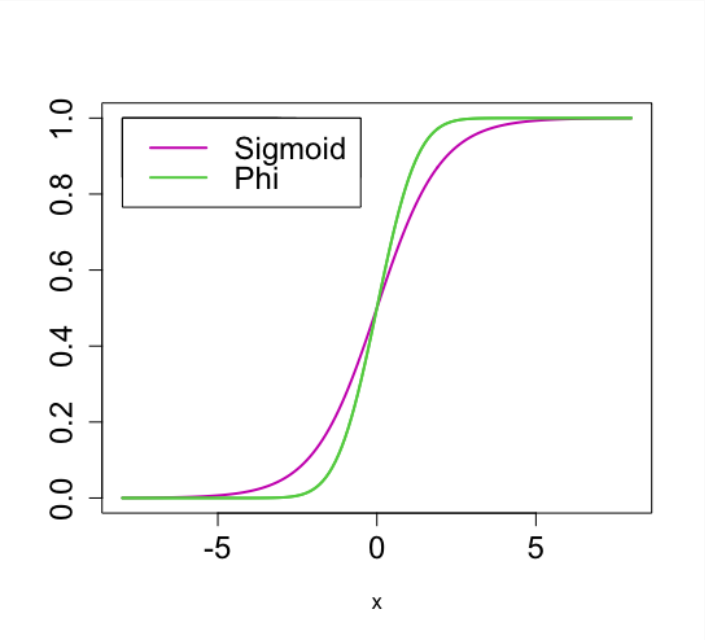
Model choices for classification vii

Other model choices

- Other link functions g are possible resulting in different models
- In principle, any function $g : \mathbb{R} \to [0, 1]$ is suitable (for modellling probabilities)
- Another common choice is the distribution function of the standard Gaussian distribution

Model choices for classification viii

$$\Phi(z) = \mathbb{P}(\mathcal{N}(0,1) \leq z)$$



yielding the so-called **probit classification model**

Properties of logistic regression i

Interpretation in the logistic model

- The sigmoid choice has the following nice interpretation:

$$\log\left(\frac{\mathbb{P}(y=1|x)}{\mathbb{P}(y=-1|x)}\right) = \log\left(\frac{\mathbb{P}(y=1|x)}{1-\mathbb{P}(y=1|x)}\right) = x^\top w + b$$

  This quantity is called the **log odds**

Properties of logistic regression ii

Odds ratio

- Odds are an expression of relative probabilities
- The odds in favor of an event A is the ratio : $\frac{\mathbb{P}(A)}{\mathbb{P}(\text{not } A)}$
- In the classification context, the **odds ratio** is defined as

$$\text{OR} = \frac{\mathbb{P}(y = 1|x)}{\mathbb{P}(y = -1|x)}$$

Interpretation

Conditionally on x,

- $\text{OR} = 1$: both events $\{y = 1\}$ and $\{y = -1\}$ occur equally often
- $\text{OR} > 1$: event $\{y = 1\}$ is more probable than $\{y = -1\}$
- $\text{OR} < 1$: event $\{y = 1\}$ is less probable than $\{y = -1\}$

Properties of logistic regression iii

Interpretation in the logistic model

- The sigmoid choice has the following nice interpretation:

$$\log\left(\frac{\mathbb{P}(y=1|x)}{\mathbb{P}(y=-1|x)}\right) = \log\left(\frac{\mathbb{P}(y=1|x)}{1-\mathbb{P}(y=1|x)}\right) = x^\top w + b$$

  This quantity is called the **log odds**

- For a classifier

$$\hat{y}_t(x_{\text{new}}) = \begin{cases} 1 & \text{if } \mathbb{P}(y=1|x_{\text{new}}) \geq t \\ -1 & \text{otherwise} \end{cases}$$

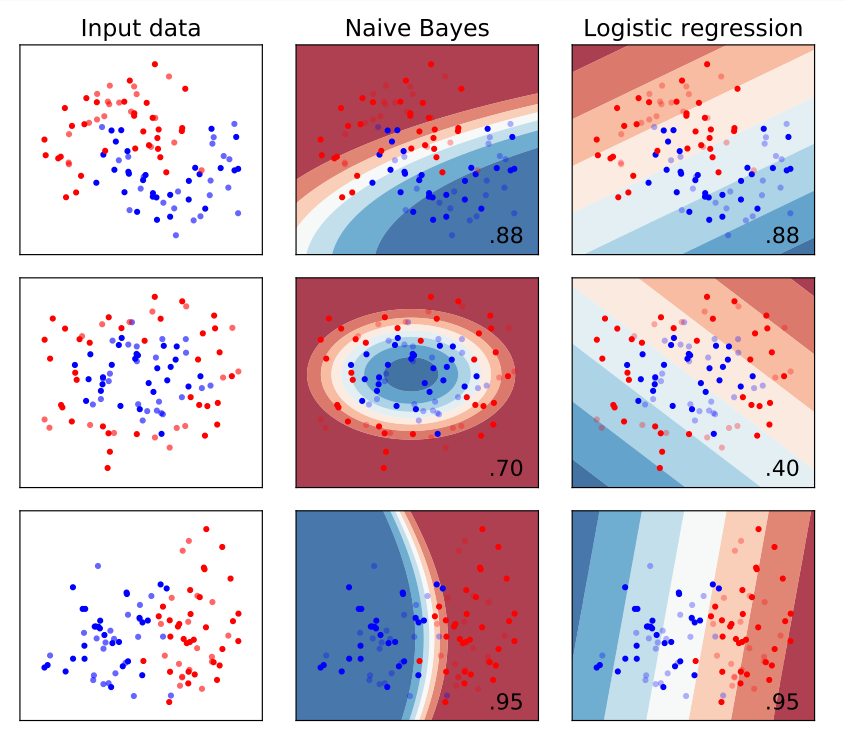
  we have for some known constant c_t

$$\hat{y}_t(x_{\text{new}}) = 1 \iff \mathbb{P}(y=1|x_{\text{new}}) \geq t \iff x_{\text{new}}^\top w + b \geq c_t$$

Properties of logistic regression iv

Interpretation of logistic regression classifier

- **Linear classification rule**
- Linear with respect to the considered features x



Outline of day 1

Training of the classifier i

Recall: Setting of logistic regression

- Model choice: $\mathbb{P}(y = 1|x) = \sigma(x^\top w + b)$
- Unknown model parameters: weights w and intercept b
- Assume data $(x_i, y_i), i = 1, \ldots, n$ to be i.i.d realizations of (X, Y)
- Compute **maximum likelihood estimators** $\hat{w}$ and $\hat{b}$

Parameter estimation

- Maximize the log-likelihood function

$$\begin{aligned}\mathcal{L}(w, b) &= \sum_{i=1}^{n} \log \mathbb{P}(Y = y_i|x_i) \\ &= \sum_{i=1}^{n} \left\{1_{y_i=1} \log(\sigma(x_i^\top w + b)) + 1_{y_i=-1} \log(1 - \sigma(x_i^\top w + b))\right\}\end{aligned}$$

- Equivalently, minimize the negative log-likelihood: $L(w) = -\mathcal{L}(w, b)$

Training of the classifier ii

General approach

- Let $\hat{y}_\theta$ be a predictor with model parameter θ
- Consider an individual **loss function** $\ell(y_i, \hat{y}_\theta(x_i))$ quantifying the loss of predicting y_i using $\hat{y}_\theta(x_i)$
- Training is performed through minimization of the total loss $L(\theta)$, i.e. the average of the individual losses:

$$\hat{\theta} \in \operatorname*{argmin}_\theta L(\theta) \qquad \text{with} \qquad L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{y}_\theta(x_i))$$

- Use an **optimization algorithm**, also called **solver**

Training of the classifier iii

Training = Optimization

- Let $L(\theta)$ be a loss function or **criterion** (e.g. -log-likelihood, classification error. . .)
- Aim: learn the model parameters θ that **optimize** the criterion (in general, minimize a loss).
- In machine learning, optimization is usually done using first-order information, i.e. gradient (derivative) of the function: $\nabla L(\theta)$

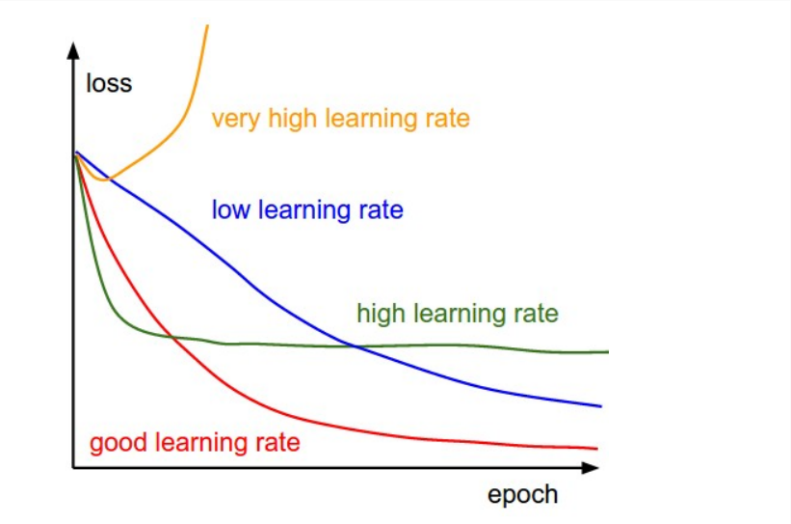
Optimization of a loss i

Simplest algorithm

- **Gradient descent** iterates

$$\theta^{t+1} = \theta^t - \lambda_t \nabla L(\theta^t)$$

where t is the iteration counter of the algorithm and λ_t is a sequence of step-sizes, also called **learning rates**.

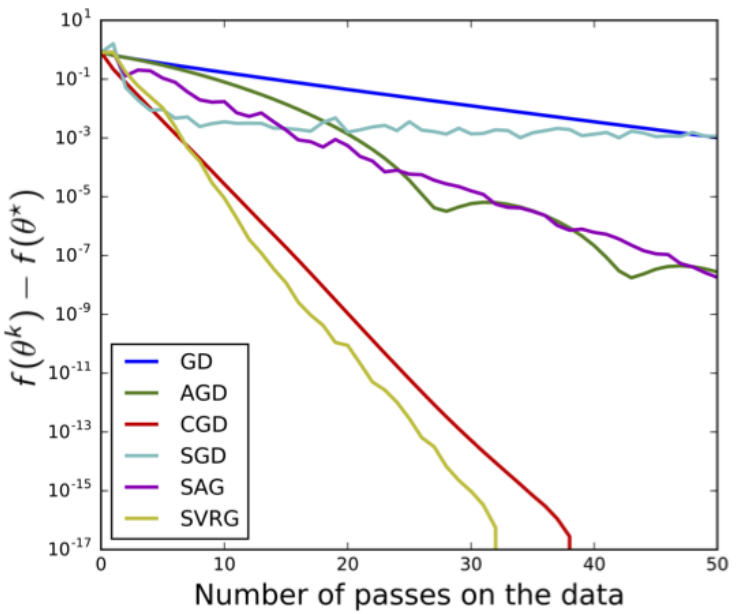


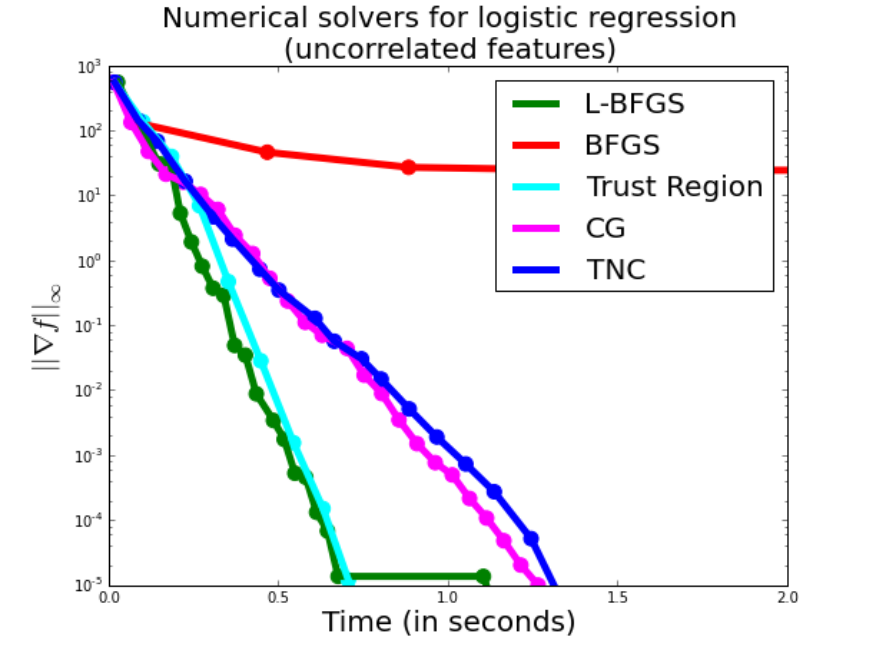
- if λ is very small, longer time to converge, computationally expensive;
- if λ is large, gradient descent may fail to converge and overshoot the minimum.

Optimization of a loss ii

Optimization of a loss iii

Many solvers out there!

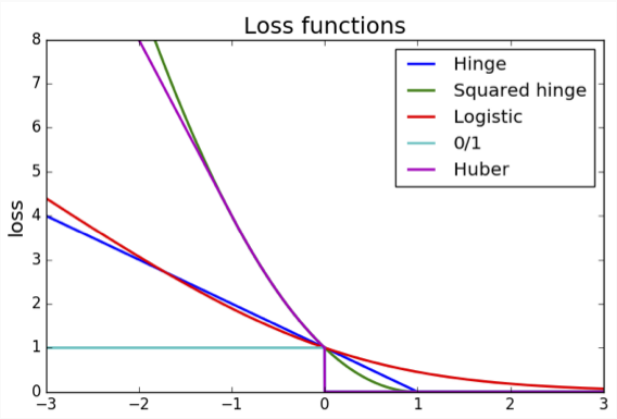




Optimization of a loss iv

Other classical loss functions for binary classification Here $\hat{y} = x^T w + b$, not predicted class label!

- Hinge loss (SVM), $\ell(y, \hat{y}) = (1 - y\hat{y})_+$
- Quadratic hinge loss (SVM), $\ell(y, \hat{y}) = \frac{1}{2}(1 - y\hat{y})_+^2$
- Huber loss $\ell(y, \hat{y}) = -4y\hat{y}\mathbf{1}_{y\hat{y}<-1} + (1 - y\hat{y})_+^2 \mathbf{1}_{y\hat{y}\geq -1}$



- These losses can be understood as a convex approximation of the 0/1 loss $\ell(y, \hat{y}) = \mathbf{1}_{y\hat{y}\leq 0}$

Outline of day 1

Non-symmetric classification errors i

Classification errors

- For each individual i in the dataset, we have
  - an actual/true label y_i
  - a predicted label $\hat{y}_i$
- **Two errors** may occur:
  1. decide $\hat{y}_i = 1$, while the truth is $y_i = -1$ (**false positive**)
  2. decide $\hat{y}_i = -1$, while the truth is $y_i = 1$ (**false negative**)
- In a practical context, the two errors are **not symmetric**, i.e. one error has much worse effects than the other
- We may wish to have a harder control on one of the errors than on the other
- In general, the labels are such that we are less tolerant to false positives than to false negatives

Non-symmetric classification errors ii

True positives $\text{TP} = \sum_{i=1}^{n} \mathbf{1}_{y_i=1,\hat{y}_i=1}$

True negatives $\text{TN} = \sum_{i=1}^{n} \mathbf{1}_{y_i=-1,\hat{y}_i=-1}$

False negatives $\text{FN} = \sum_{i=1}^{n} \mathbf{1}_{y_i=1,\hat{y}_i=-1}$

False positives $\text{FP} = \sum_{i=1}^{n} \mathbf{1}_{y_i=-1,\hat{y}_i=1}$

Confusion matrix

True y	Predicted $\hat{y}$ 1	- 1
1	TP	FN
-1	FP	TN

Errors ratios

Precision

$$\frac{\text{TP}}{\#(\text{predicted P})} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- classifier's ability to not declare positive a negative individual
- false-discovery rate FDR $= 1 -$ precision

Recall

$$\frac{\text{TP}}{\#(\text{real P})} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- ability to detect all the positives
- recall $=$ sensitivity

Accuracy

$$\frac{\text{TP} + \text{TN}}{n}$$

F-Score

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Varying the threshold

Recall: Definition of a classifier

- A **classifier** is defined by

$$\hat{y}_t(x_{\text{new}}) = \begin{cases} 1 & \text{if } \widehat{P}(y = 1|x_{\text{new}}) \geq t \\ -1 & \text{otherwise} \end{cases}$$

  for some **threshold** $t \in (0, 1)$ selected by the user
- Varying threshold t gives rise to a whole family of classifiers (all based on the same model for $\mathbb{P}(y = 1|x)$)
- Error rates vary with threshold t
- In practice, choose threshold t according to **personal preferences** for acceptable error rates

Towards error-ratio curves

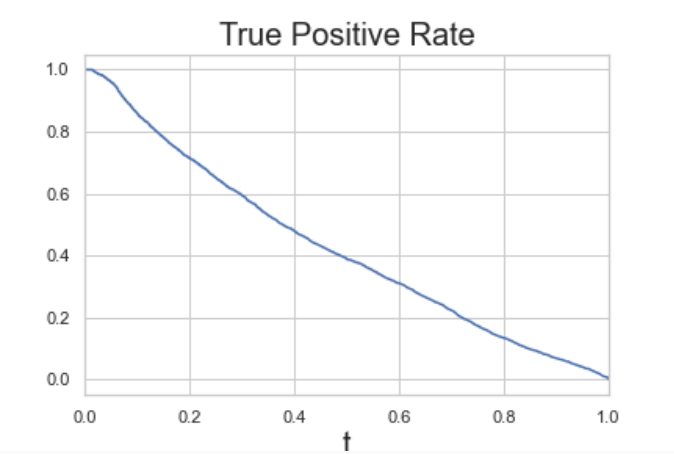
True and false positive rate

- True positive rate (= recall):

$$\mathrm{TPR}_t = \frac{\mathrm{TP}_t}{\mathrm{TP}_t + \mathrm{FN}_t}, \quad t \in [0, 1]$$

- False positive rate (= 1 - specificity):

$$\mathrm{FPR}_t = \frac{\mathrm{FP}_t}{\mathrm{FP}_t + \mathrm{TN}_t}, \quad t \in [0, 1]$$



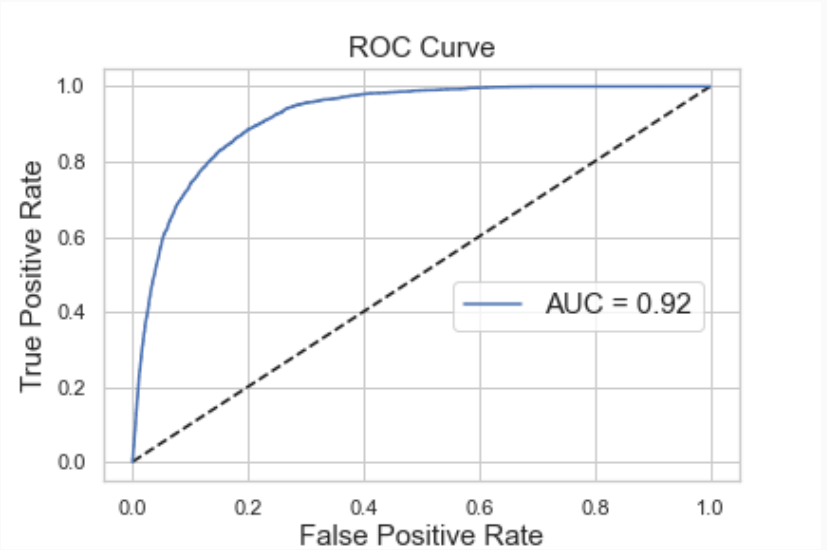
ROC curve

- The **ROC curve** (Receiver Operating Characteristic) is the curve with coordinates

  $(\text{FPR}_t, \text{TPR}_t) \quad \text{for } t \in [0, 1]$

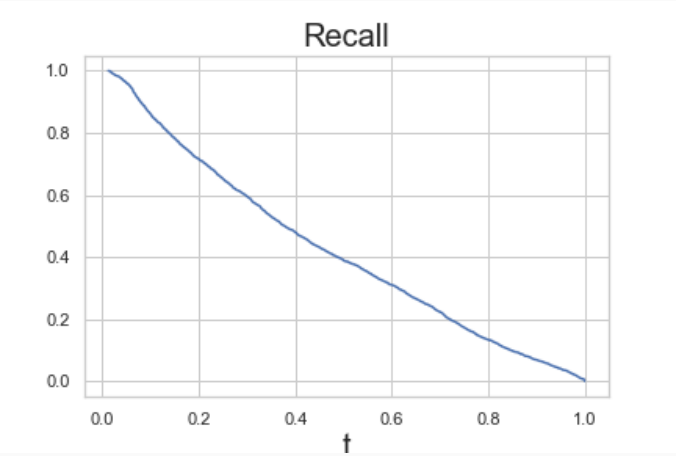
- The **AUC-ROC score** is the Area Under the ROC curve
- The larger the AUC, the better
- A classifier whose ROC curve is uniformly larger than the ROC curve of another classifier is preferable

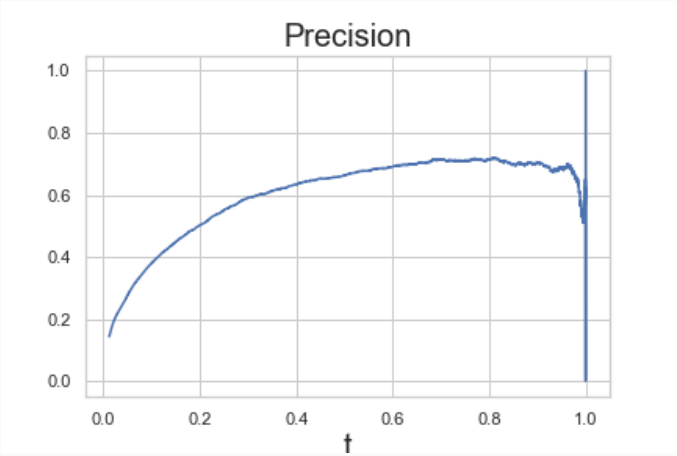
- The line $y = x$ represents the ROC curve of the family of **random classifiers** that output $\hat{y}_i = 1$ at random with proba $t \in [0, 1]$



Precision-recall curve i

- **Recall**$_t = \frac{\mathrm{TP}_t}{\mathrm{TP}_t + \mathrm{FN}_t} = \mathrm{TPR}_t$ (decreasing in t)
- **Precision**$_t = \frac{\mathrm{TP}_t}{\mathrm{TP}_t + \mathrm{FP}_t}$ (not necessarily monotone)



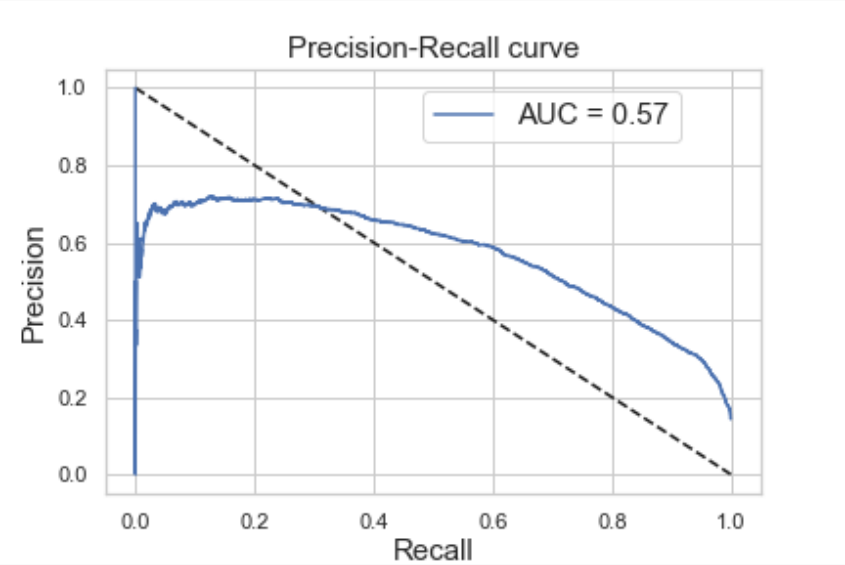


Precision-recall curve ii

- The **precision-recall curve** is the curve with coordinates

$$(\text{Recall}_t, \text{Precision}_t) \quad \text{for } t \in [0, 1]$$

- The larger the curve (or the area under the curve, i.e. the **AUC-PR score**), the better



Test set to compute performance metrics

Unbiased performance assessment

- In general, any performance measure gives better results on the data that served to train the classifier than on new data
- Always use "fresh" data, called **test set** or **hold-out datatest**, to obtain an unbiased evaluation of the final classifier

Training and test sets

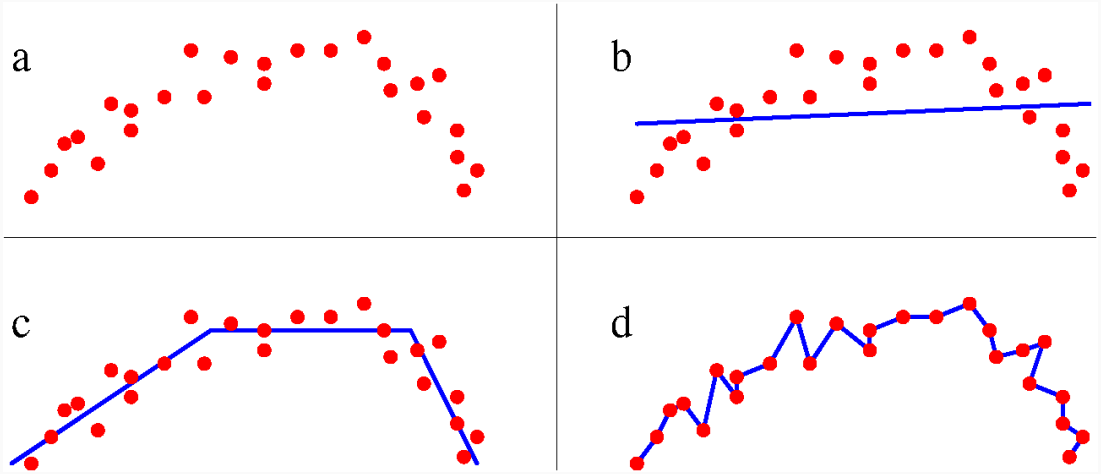
- Split data $\mathcal{D} = \{(y_i, x_i), i = 1, \ldots, n\}$ into a training set and a test set: $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}} = \mathcal{D}$
  - Use $\mathcal{D}_{\text{train}}$ to learn the classifier
  - Use $\mathcal{D}_{\text{test}}$ to evaluate error metrics and performance measures
- The split is done in a random fashion
- Typical proportions for training and test set: 70-30, 80-20, 50-50

Outline of day 1

Overfitting and model selection i

Problem of overfitting

- Complex models have a huge number of model parameters
- With a given dataset, only a limited number of parameters can be estimated accurately
- If the model is not complex enough, the result is **biased**
- If the model is too complex, the result is unstable and **highly variable**



Overfitting and model selection ii

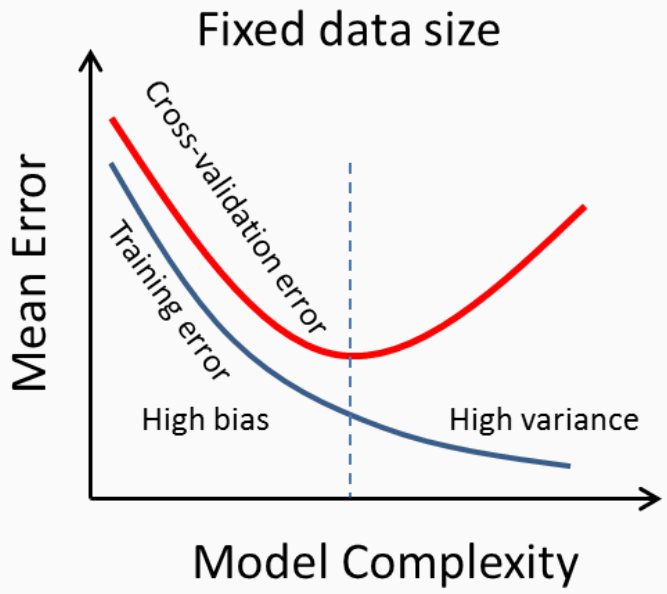
Problem of overfitting

- Often many features $x_i = (x_{i,1}, \ldots, x_{i,d}) \in \mathbb{R}^d$ are observed: d **large**
- Features may be **correlated**
- Not all features are useful for the prediction of label y
- The more features, the **more model parameters** to estimate

Overfitting and model selection iii

Model selection problem

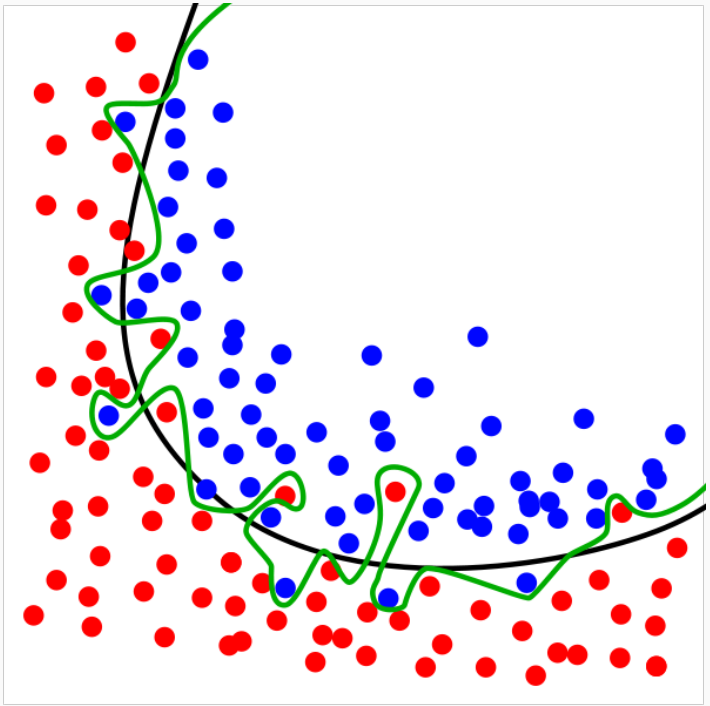
- How many features? How many parameters?
- Which features are relevant for predicting y?
- How many parameters can be accurately learned with the given data?



Overfitting and model selection iv

Goal

- **Generalization**: A trained classifier has to be generalizable, it must be able to work on other data than the training dataset
- Generalizable means "works without **overfitting**"
- no feature engineering
- we want an automatic solution, a **data-driven** or **adaptive** selection rule



Penalization i

Logisitic regression

- $\mathbb{P}(y = 1|x) = \sigma(x^\top w + b)$
- $w_k = 0$ means that the k-th feature is not relevant for prediction

Penalization

- Ideas:
  - Either set some coefficients w_k to 0 in an automatic way (**model selection**)
  - Or penalize the use of large coefficients w_k (**regularisation**)
- Technically: modifiy the criterion by adding a **penalty term** to the loss function

Penalization ii

Standard logisitic regression

- Loss function: $L(w, b) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_{w,b}(x_i))$
- Best classifier $\hat{y}_{\hat{w},\hat{b}}$ has model parameters $\hat{w}$ and $\hat{b}$ such that

$$L(\hat{w}, \hat{b}) = \min_{w,b} L(w, b)$$

- Adding more features x_k to the model decreases the loss at its optimum:

$$L(\hat{w}_{\text{many features}}, \hat{b}) < L(\hat{w}_{\text{small model}}, \hat{b})$$

- The loss quantifies the **goodness of the fit** of the model to the data

Penalization iii

Penalized logisitic regression

- A **penalty** is a function $\text{pen}(w)$ taking "large" values for "large" values of w
- To avoid overfitting consider the classifier $\hat{y}_{\tilde{w},\tilde{b}}$ with parameters $\tilde{w}, \tilde{b}$ that minimize

$$L(w, b) + \frac{1}{C}\,\text{pen}(w)$$

- Constant $C > 0$ is a **tuning** or **smoothing** parameter, that helps to **balance** goodness-of-fit and penalization

Form of the penalty i

Choice of the penalty

- Different penalties with different properties exist

Ridge- or ℓ_2-penalty

- The **ridge** penalization considers

$$\text{pen}(w) = \|w\|_2^2 = \sum_{j=1}^{d} w_j^2$$

- Penalization of the "magnitude" of w, **shrinkage** of coefficients
- This is the most widely used penalization
- It helps with correlated features
- It helps training by making the problem more convex

Form of the penalty ii

ℓ_1-penalty

- The ℓ_1-penalty is given by

$$\text{pen}(w) = \|w\|_1 = \sum_{j=1}^{d} |w_j|$$

- **ℓ_1-penalized logistic regression**
- The ℓ_1-penalty induces **sparsity**, i.e. some coefficients w_k are set to 0
- Leads to a simpler model, with reduced dimension
- In the context of linear regression (not logistic), the ℓ_1-penalty gives what is called the **Lasso** estimator.

Form of the penalty iii

Combinations

- Several penalties may be combined
- The **elastic-net** uses the penalty

$$\text{pen}(w) = \|w\|_2^2 + \alpha\|w\|_1$$

- It has the advantages of both ridge- and ℓ_1-penalization
- $\alpha \geq 0$ balances the two penalizations

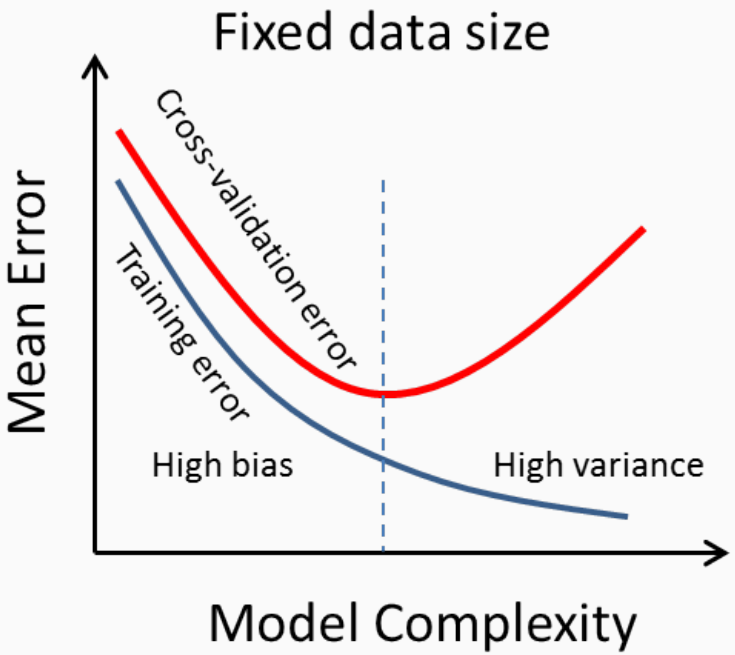
Penalization constant C

Minimization of the penalized loss

$$\min_{w,b} L(w, b) + \frac{1}{C} \text{pen}(w)$$

Impact of penalization constant C

- Depending on C the fitted model is more or less complex
- C defines the balance between goodness-of-fit and overfitting
- Choose C such that the model generalizes well



Cross validation to choose the penalization constant i

Cross validation

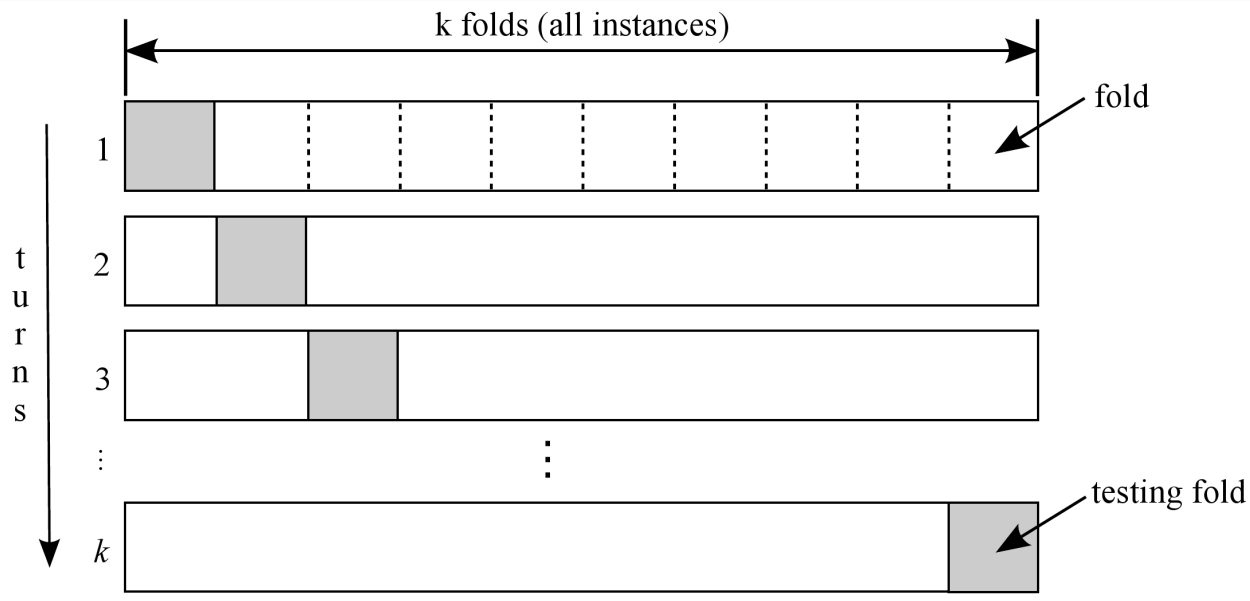
- Use cross validation to choose $C > 0$
- Cross validation is a resampling method that repeatedly uses different portions of the data to train and validate a model
- There is **no machine learning without cross validation** at some point!

Cross validation to choose the penalization constant ii

K-fold cross validation

- Most standard cross-validation technique
- Consider a set $\mathcal{C} = \{C_1, \ldots, C_J\}$ of **candidate values** for constant C
- Choose a validation criterion or **score** to be optimized (accuracy, AUC-ROC...)
- Partition data into K subsets $\mathcal{D}_1, \ldots, \mathcal{D}_K$ of equal size
- Typical values are $K = 5$ and $K = 10$

Cross validation to choose the penalization constant iii



Cross validation to choose the penalization constant iv

K-fold cross-validation algorithm

1. `For` $k = 1, \ldots, K$ `do:`
   - Select one subset to be the current **validation set**: $\mathcal{D}_{\text{valid},k} = \mathcal{D}_k$ and the remaining data $\mathcal{D}_{\text{train},k} = \cup_{\ell \neq k} \mathcal{D}_\ell$ as current **training set**
   - `For` $j = 1, \ldots, J$ `do`
     - Train the penalized model **with constant** C_j on $\mathcal{D}_{\text{train},k}$ yielding a predictor $\hat{y}_{\hat{w}_{j,k}}$ with

$$\hat{w}_{j,k} \in \operatorname*{argmin}_w \left\{ \frac{1}{n_{\text{train}}} \sum_{i \in \mathcal{D}_{\text{train},k}} \ell(y_i, \hat{y}_w) + \frac{1}{C_j} \text{pen}(w) \right\}$$

     - Evaluate score on the **validation set** $\mathcal{D}_{\text{valid},k}$:

$$S_{j,k} = \text{score}(\hat{y}_{\hat{w}_{j,k}}, \mathcal{D}_{\text{valid},k})$$

Cross validation to choose the penalization constant v

2. Select **best constant** $\hat{C}$ that optimizes the score over all runs:

$$\hat{C} = C_{j_{\text{best}}} \quad \text{with} \quad j_{\text{best}} = \underset{1 \leq j \leq J}{\text{argmax}} \frac{1}{K} \sum_{k=1}^{K} S_{j,k}$$

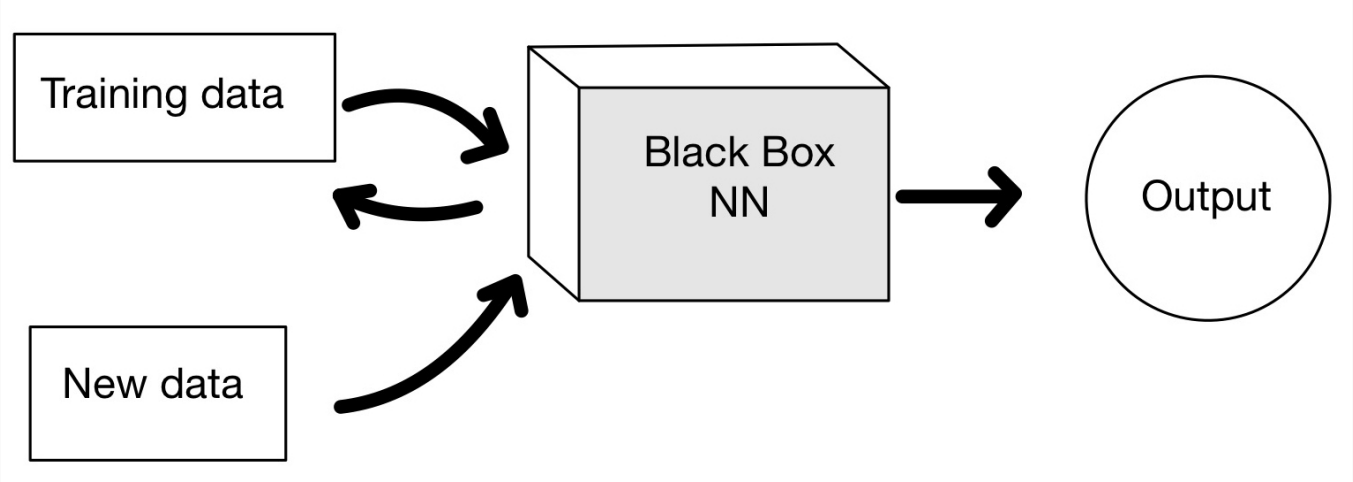
3. Train the **final classifier** $\hat{y}_{\hat{w}_{\text{final}}}$ on the **whole dataset** $\mathcal{D}$ with penalization constant $\hat{C}$:

$$\hat{w}_{\text{final}} \in \underset{w}{\text{argmin}} \left\{ \frac{1}{n} \sum_{i \in \mathcal{D}} \ell(y_i, \hat{y}_w) + \frac{1}{\hat{C}} \text{pen}(w) \right\}$$

4. To **evaluate the performance** (error rates, ROC curve) of $\hat{y}_{\hat{w}_{\text{final}}}$ use "fresh" data, i.e. a test set $\mathcal{D}_{\text{test}}$ that has not been used during the cross-validation procedure

Outline of day 1

The general picture i



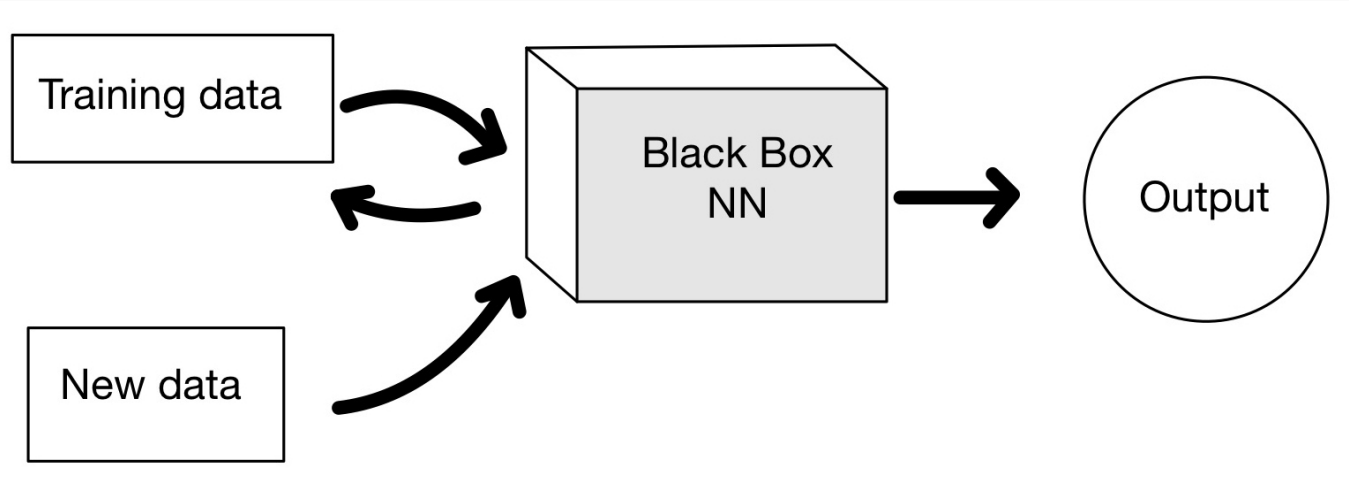
The general picture ii

Ingredients for a deep neural network

- Input training data
  - supervised setting: $(x_i, y_i), 1 \leq i \leq n$
  - unsupervised setting: $x_i, 1 \leq i \leq n$
- NN-Black Box is defined of
  - an **architecture** $\hookrightarrow$ chosen by the user
  - **parameters** $\hookrightarrow$ learnt on training data by minimizing a loss
- **Loss** for training the model:
  - supervised setting: prediction accuracy on training data
  - unsupervised setting: cross entropy (negative log-likelihood) of training data
- With the trained model, for any **new** observation x^{new}:
  - supervised setting: the NN predicts an output y^{pred}
  - unsupervised setting: the NN computes a representation of x^{new}

Architecture

What's in the box ?



Artificial neuron: an example i

Example: Logistic regression

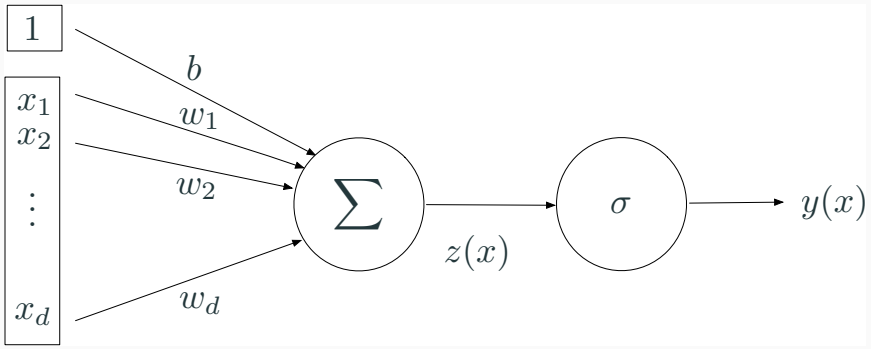
- Dataset $(x_1, y_1), \ldots, (x_n, y_n)$ with $y_i \in \{0, 1\}$ and $x_i = (x_i^1, \ldots, x_i^d) \in \mathbb{R}^d$
- Given a vector of input features $x \in \mathbb{R}^d$, consider

$$\mathbb{P}(Y = 1 | X = x) = \sigma(\langle w, x \rangle + b) = \frac{e^{\langle w, x \rangle + b}}{e^{\langle w, x \rangle + b} + 1}$$

- Model **weights** $w \in \mathbb{R}^d$ and intercept (or **bias**) $b \in \mathbb{R}$

Artificial neuron: an example ii

Graphical representation of a logistic regression



Deep learning vocabulary

- $x = (x^1, \ldots, x^d)$ is the **input** (a single feature vector)
- $z(x) = \langle w, x \rangle + b$ is called **pre-activation**
- $y(x) = \sigma(z(x))$ is the output (in $[0, 1]$ in this case)
- $w = (w^1, \ldots, w^d)$ are **weights** and b is **bias**
- σ is an **activation** function

This is called a **unit** or an **artificial neuron**

Artificial neuron: an example iii

Every artificial neuron

- takes a vector as input
- applies a **linear** mapping with weights that are to be learned
- applies a **nonlinear** known activation function
- outputs a single value

Network layer: an example i

Softmax regression or Multinomial logistic regression

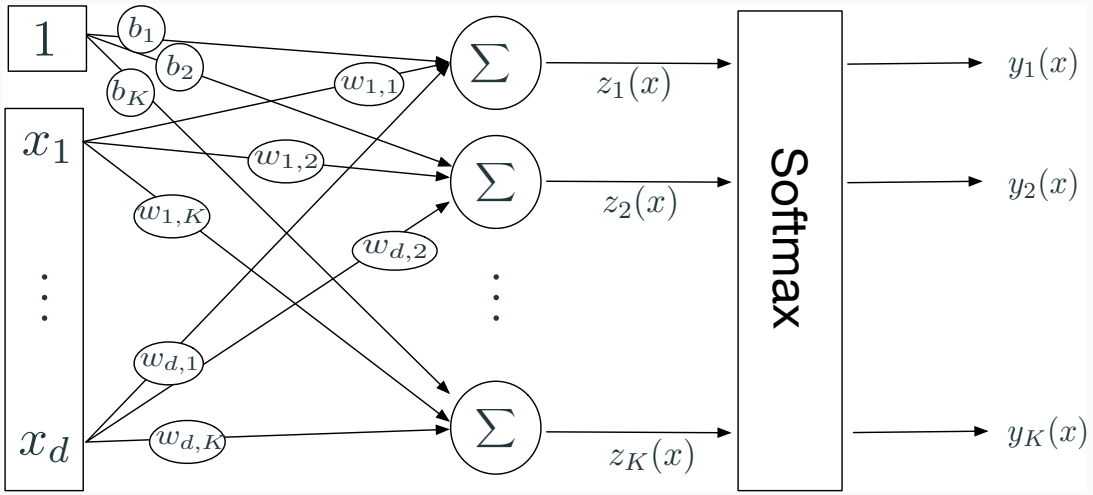
- Generalization of logistic regression to more than two classes
- Dataset $(x_1, y_1), \ldots, (x_n, y_n)$ with $y_i \in \{1, \ldots, K\}$ and $x_i = (x_i^1, \ldots, x_i^d) \in \mathbb{R}^d$
- Given an input features vector $x \in \mathbb{R}^d$, consider

$$\mathbb{P}(Y = \mathbf{k}|X = x) = \frac{e^{\langle \mathbf{w_k}, x \rangle + \mathbf{b_k}}}{\sum_{l=1}^{K} e^{\langle w_l, x \rangle + b_l}}$$

  for $k \in \{1, \ldots, K\}$
- One vector of weights $w_k \in \mathbb{R}^d$ for each class k
- Stack all weights in a $d \times K$ matrix $\boldsymbol{W} = [w_1, \ldots, w_K]$ and $\boldsymbol{b} = (b_1, \ldots, b_K)$

Network layer: an example ii

Softmax regression



- input: x
- output: $y_1(x), \ldots, y_K(x)$
- pre-activations or logits: $z_k(x) = \langle w_k, x \rangle + b_k$
- **Softmax activation**: $y_k(x) = \frac{e^{z_k(x)}}{\sum_{l=1}^{K} e^{z_l(x)}}$
  Matrix notation: $y(x) = \text{softmax}(z(x)) = \text{softmax}(\boldsymbol{W}^\top x + \boldsymbol{b})$

- This is called a **layer**
- A layer is composed of multiple neurons, also referred to as nodes
- Softmax is often the **output** layer in classification tasks

Network layer: an example iii

Every layer

- is composed of **multiple neurons**
- applied to the same input vector
- the layer's output is the vector with
  - the values produced by the neurons or
  - the result of a function applied to those values
- the number of neurons is referred to the **width** of the layer
- A layer where **all** input neurons are connected to **all** output neurons is called **dense** or **fully connected**

Network training: an example

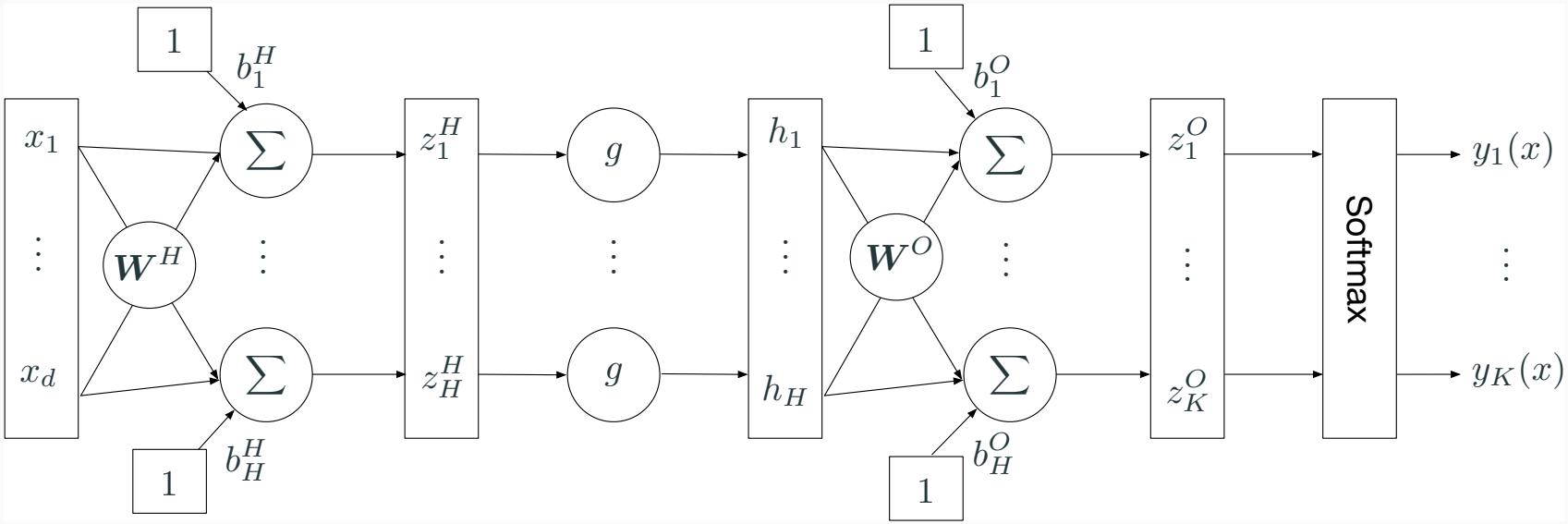
Softmax regression

- Model parameters: $\boldsymbol{W}, \boldsymbol{b}$
- To be learnt from the training data
- Training by **optimization of a criterion** or loss function
- Here: cross entropy (negative log-likelihood)

$$\ell(\boldsymbol{W}, \boldsymbol{b}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}_{y_i=k} \log \left(\frac{e^{\langle w_k, x \rangle + b_k}}{\sum_{l=1}^{K} e^{\langle w_l, x \rangle + b_l}} \right)$$

Neural network: an example i

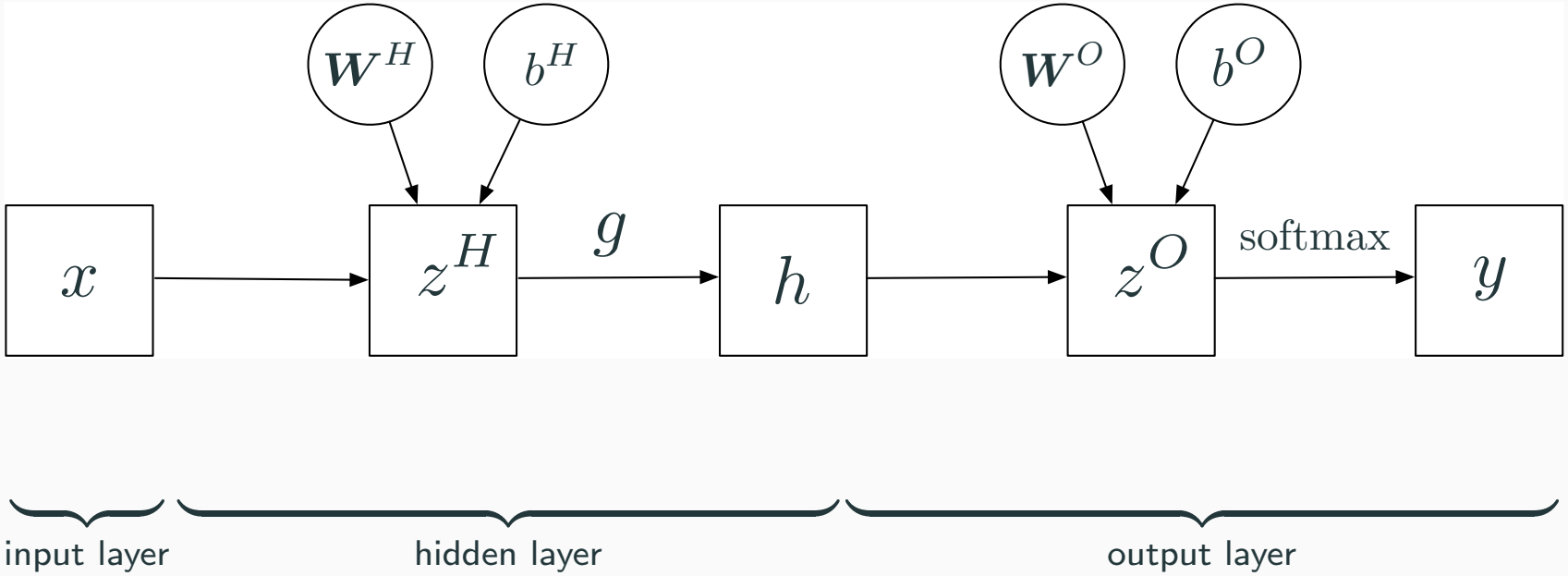
Softmax regression with one hidden layer



- g is a nonlinear **activation function** applied entrywise on $z_1^H, \ldots, z_H^H$
- The **width** of the first layer is H

Neural network: an example ii

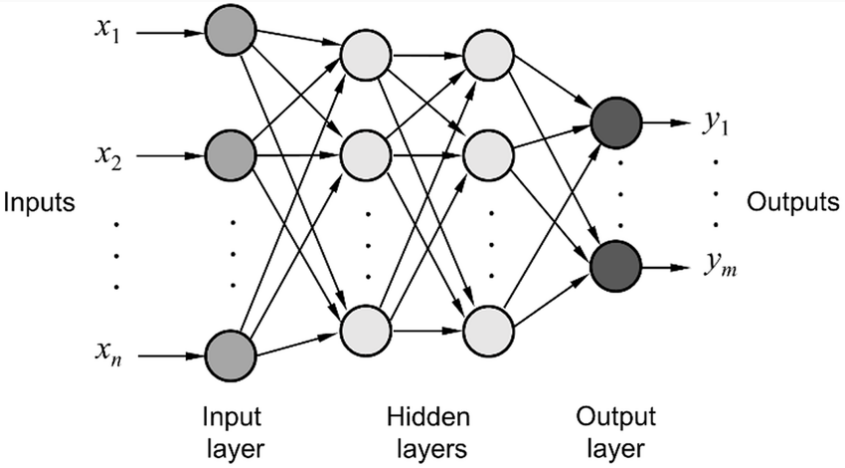
Alternative representation of the same network



$$
\begin{aligned}
y(x) &= \text{softmax}(z^{(O)}) = \text{softmax}(\boldsymbol{W}^{(O)\top} h + \boldsymbol{b}^{(O)}) \\
&= \text{softmax}(\boldsymbol{W}^{(O)\top} g(z^{(H)}) + \boldsymbol{b}^{(O)}) \\
&= \text{softmax}(\boldsymbol{W}^{(O)\top} g(\boldsymbol{W}^{(H)\top} x + \boldsymbol{b}^{(H)}) + \boldsymbol{b}^{(O)})
\end{aligned}
$$

Neural network: an example iii

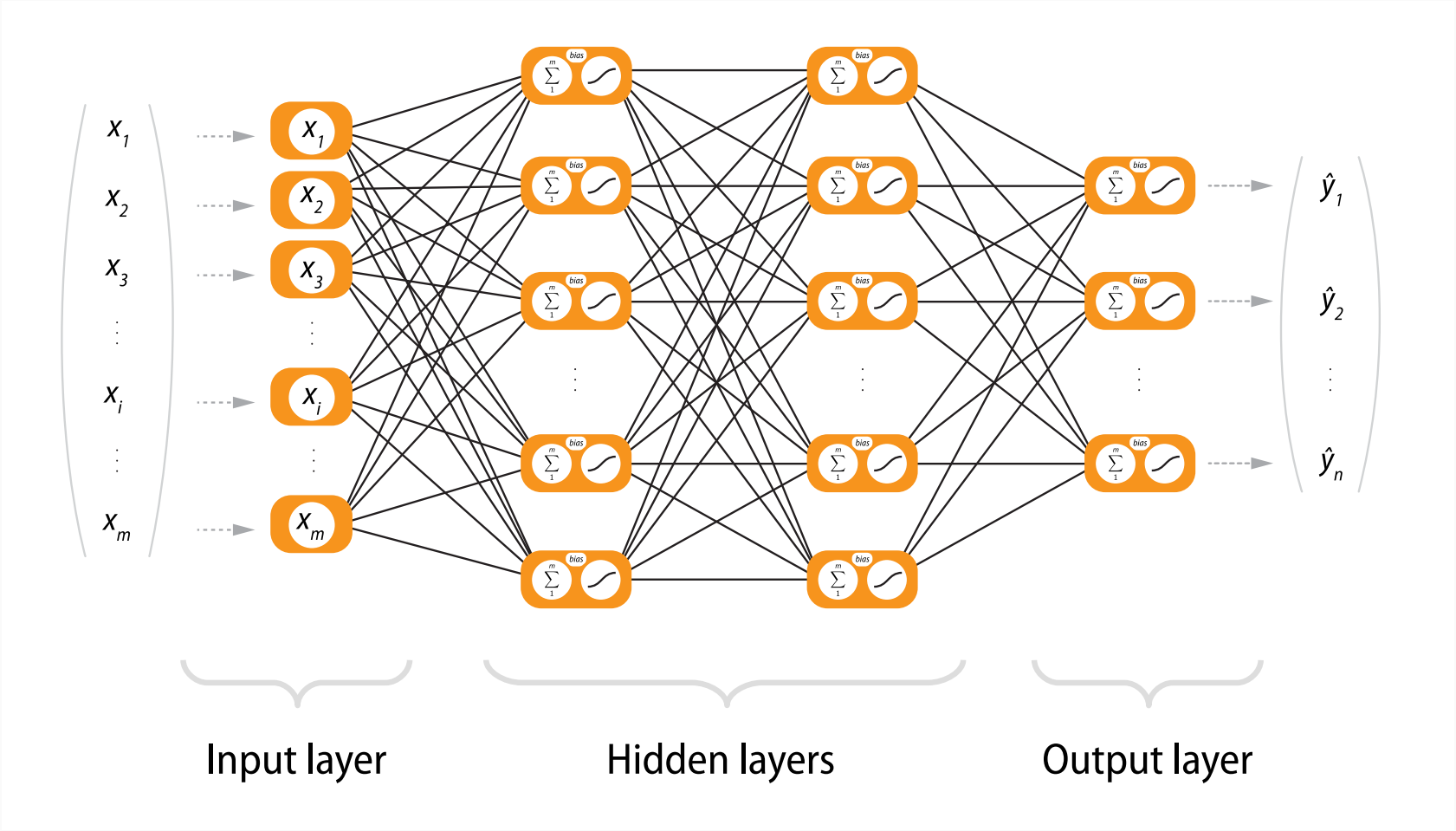
Neural network with several hidden layers



A neural network

- is composed of multiple layers: an **input layer**, **hidden layers** and an **output layer**
- the **depth** of a neural network is the number of layers
- each layer has a specific **width**

The architecture of a FFNN i



[From FIDLE `https://fidle.cnrs.fr`]

Neural networks for function approximation

- Aim: **build an approximation** f of some function f^* that maps an input x to an output y, i.e. $y = f^*(x)$
- The neural network is the function f
- Layers correspond to functions that are composed in some order: $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$ has 3 layers
- The network describes how the functions are composed together (**computational graph**)
- $f^{(1)}$ is the first layer, while $f^{(3)}$ is the last one
- Each layer $f^{(i)}$ is the composition of
  - a **linear function**: $u \mapsto \boldsymbol{W}^{(i)\top} u + b^{(i)}$ and
  - a nonlinear **activation function** $g^{(i)}$ applied entrywise

Mathematical result

Function approximation

- The **universal approximation theorem** states that a basic NN is a universal approximator
- Roughly, if there is no limit on the widths of the layers, for any function f^*, there exists a NN f that approximates f^* with desired accuracy
- This also holds for shallow networks when the number of hidden units is sufficiently large
- **Nonlinear activations** functions are key

Designing a neural network architecture i

Questions

- how many layers?
- width of the different layers?
- **shallow versus deep** neural networks

Shallow neural networks

- on **simple problems** a single layer can definitely get the job done
- on more challenging tasks one might not reach acceptable levels of accuracy
- generally need more units per layer and so more parameters to accommodate the complexity
- **easier to optimize**

Designing a neural network architecture ii

Deeper neural networks

- can get the job done with **fewer units per layer** and far fewer parameters
- leads to **good generalizations** on a test set
- can be difficult to train

Historically

- First neural networks were explored with shallow architectures and performance was unsatisfactory
- The breakthrough of neural networks came when deeper architectures were devised

This afternoon: practical session

- Logisitic regression

Thank you!