Introduction to Machine Learning and Deep Learning with Python

Day 1

Tabea Rebafka

CNRS Formation Entreprise
28 – 30 June, 2023, Paris

Agenda I

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

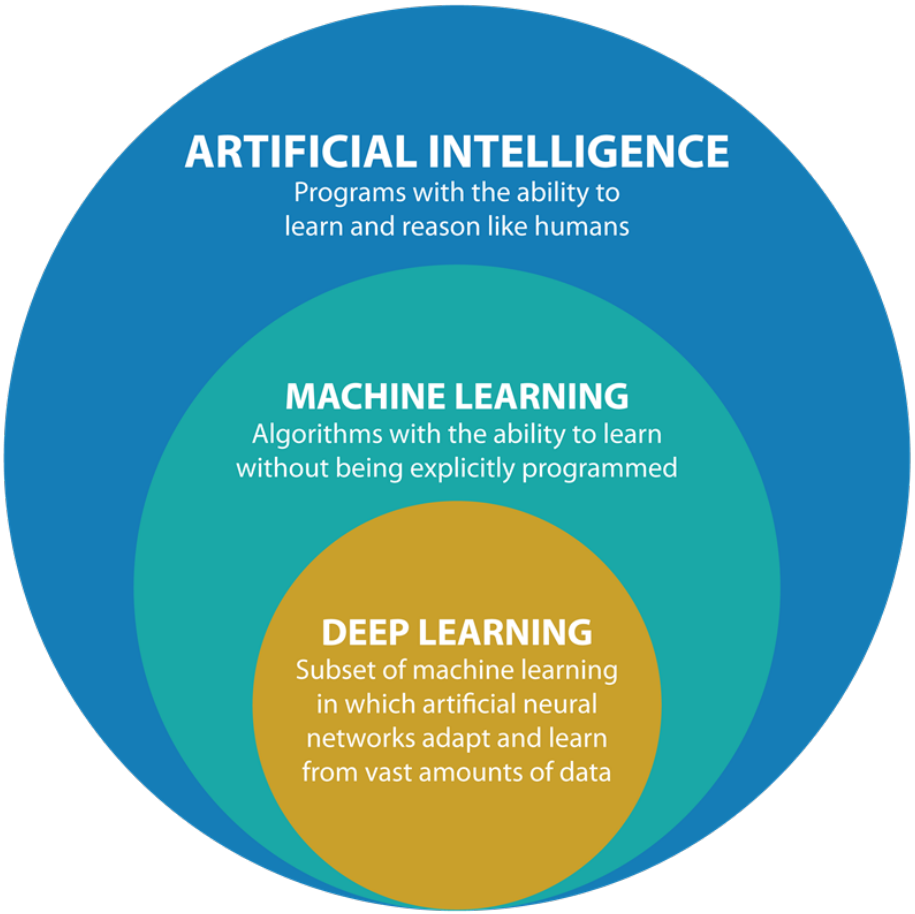
Outline of day 1

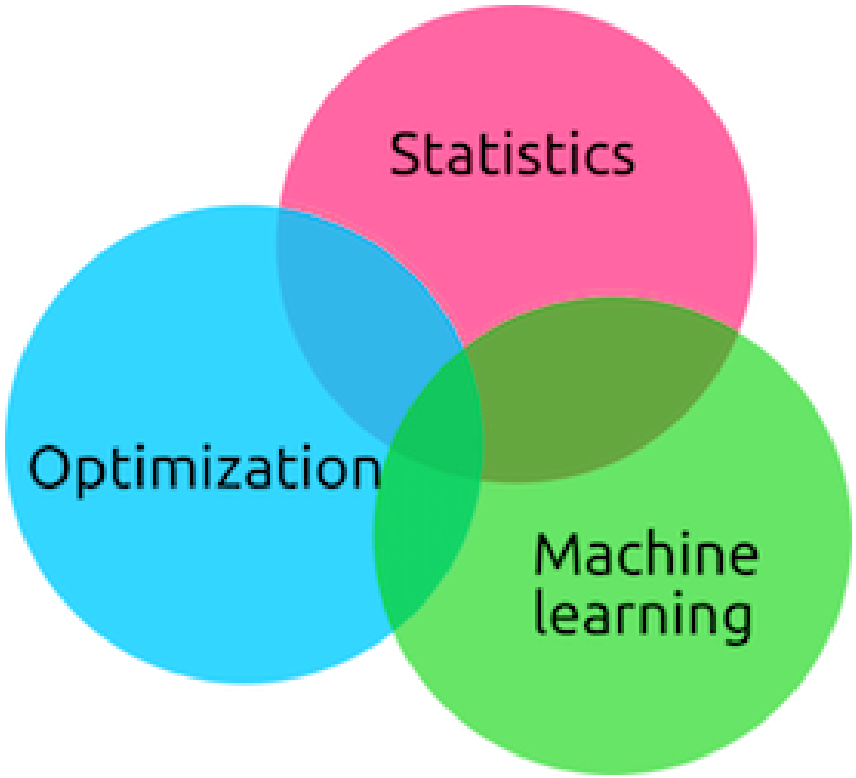
1. Introduction to this course
2. Binary classification
3. Logistic regression
4. Training of the classifier
5. Performance metrics
6. Overfitting, penalization and cross validation

Outline of day 1

1. Introduction to this course
2. Binary classification
3. Logistic regression
4. Training of the classifier
5. Performance metrics
6. Overfitting, penalization and cross validation

General picture

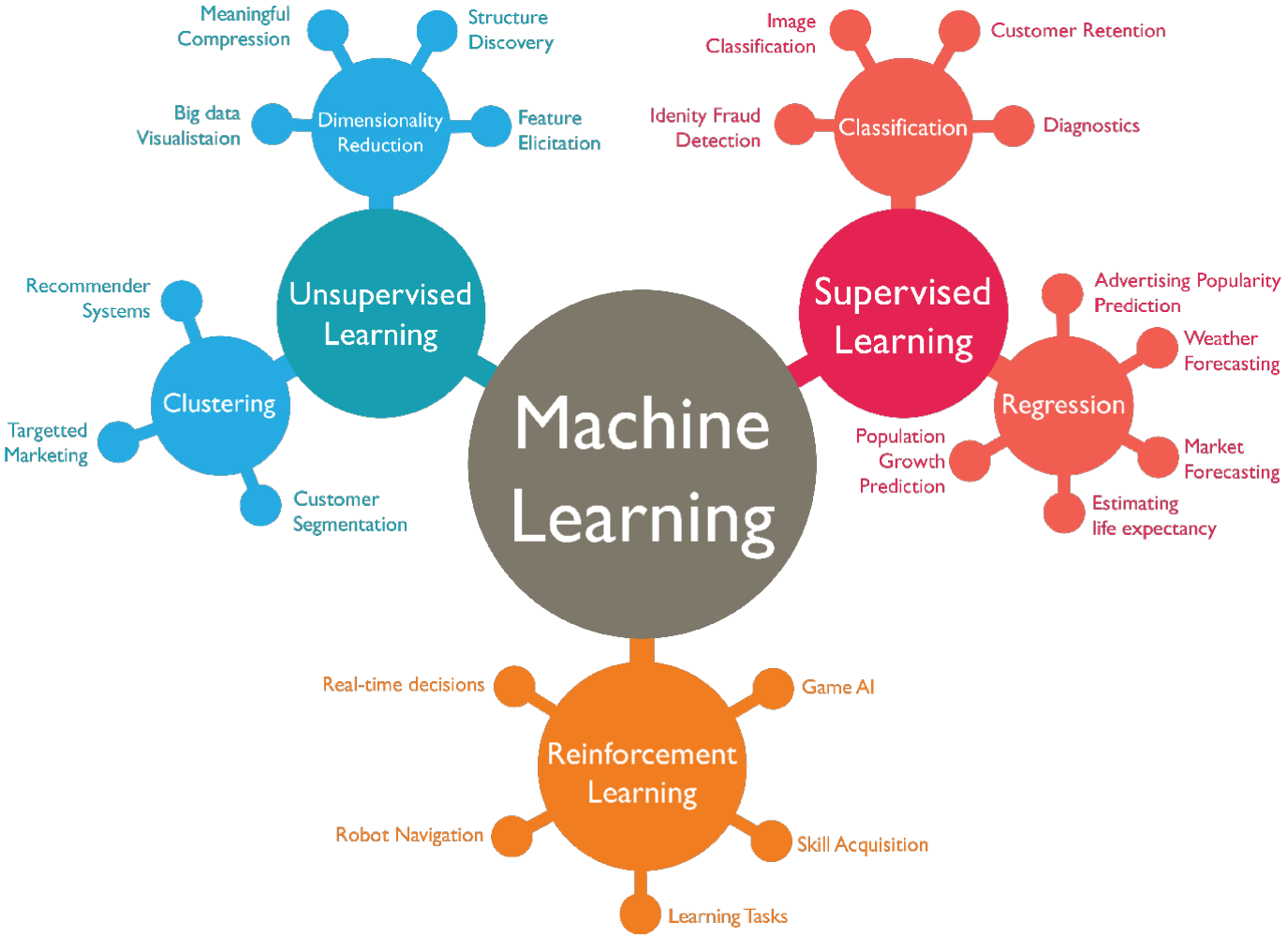




Data analysis for big data

- 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

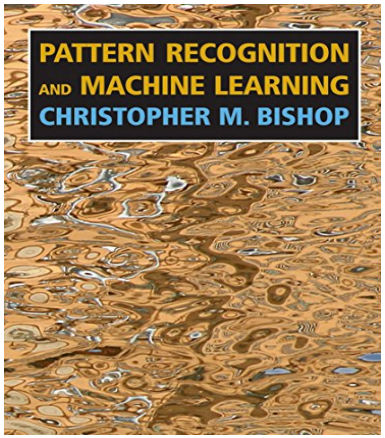
API	Pandas	Scikit-Learn	PyTorch	Tensorflow	Keras[1]
Usage	Stats	ML and stats	ML & DL	DL	DL
Who?		academic world		Google	Google
API level	High		High	Low	High

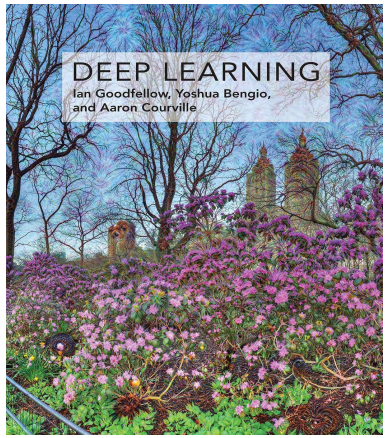
- High-level APIs are fairly abstracted, meaning that they are more generic and therefore limited in functionality. High-level API are easier to use.
- Low-level APIs are much more detailed and specific due to a low level of abstraction. Low-level APIs allow for finer control over application functions.

[1] Part of Tensorflow since 2017

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/`

Others

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

Outline of day 1

1. Introduction to this course
2. Binary classification
3. Logistic regression
4. Training of the classifier
5. Performance metrics
6. Overfitting, penalization and cross validation

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 $y_{\text{new}} \in \{-1, 1\}$

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 $\{-1, 1\}$:

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

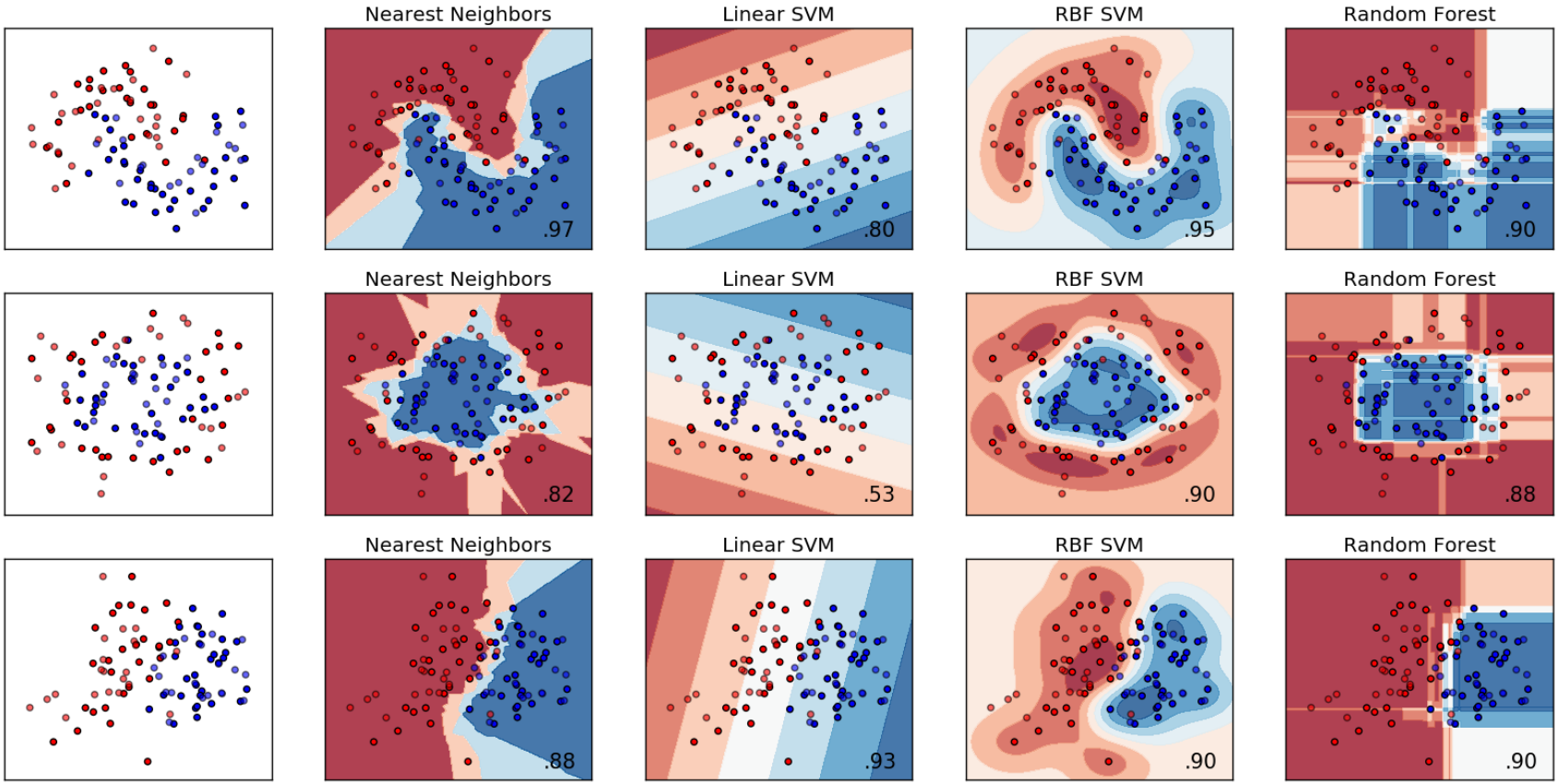
- $\hat{y}$ is learned on the data, that is, its form (or parameters) are **trained** on the historical data $\{(x_i, y_i), i = 1, \ldots, n\}$ containing observed labels
- For any new feature vector x_{new}, we can **make a prediction** of its label by evaluating the classifier: $\hat{y}(x_{\text{new}})$

Outline of day 1

1. Introduction to this course
2. Binary classification
3. Logistic regression
4. Training of the classifier
5. Performance metrics
6. Overfitting, penalization and cross validation

Construction of a classifier

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



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

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

Goal

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

Generalized linear models

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

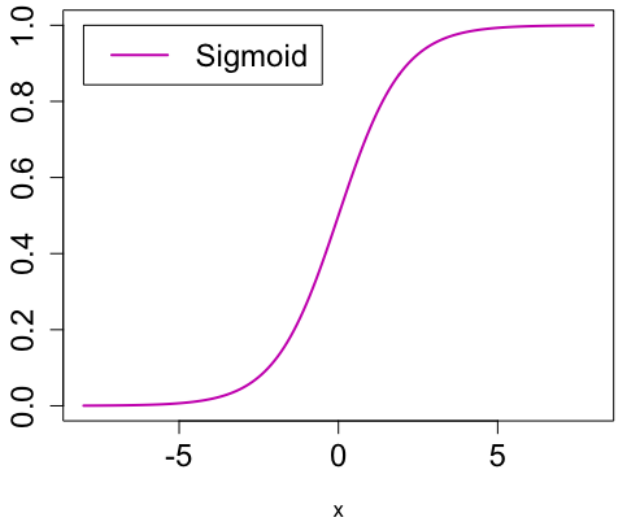
$$\mathbb{P}(y = 1|x) = f(x^\top w + b)$$

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

Logistic regression model or logit model

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

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



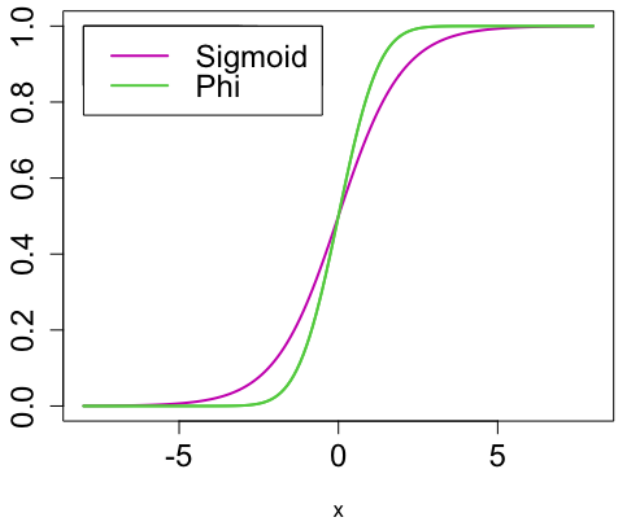
- Thus, the logistic regression model (or logit model) is defined as

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

Other model choices

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

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



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

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

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 $\{y = -1\}$

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

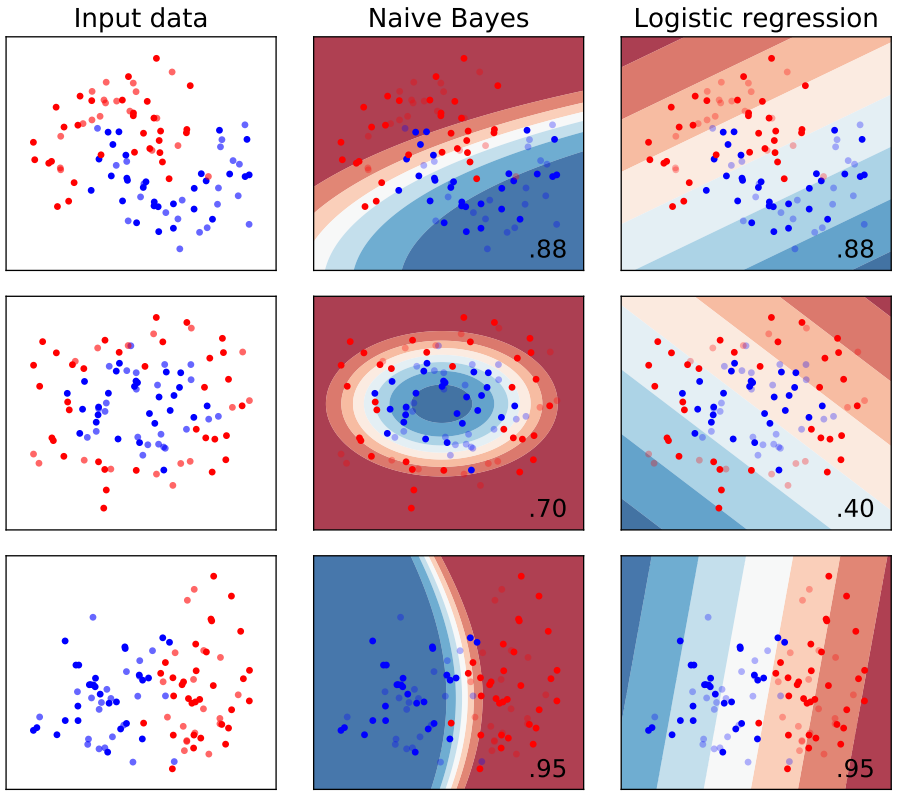
$$\hat{y}_t(x_{\text{new}}) = \begin{cases} 1 & \text{if } \mathbb{P}(y=1|x_{\text{new}}) \geq t \\ -1 & \text{otherwise} \end{cases}$$

  we have for some known constant c_t

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

Interpretation of logistic regression classifier

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



Outline of day 1

1. Introduction to this course
2. Binary classification
3. Logistic regression
4. Training of the classifier
5. Performance metrics
6. Overfitting, penalization and cross validation

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

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

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

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:

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

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

Training = Optimization

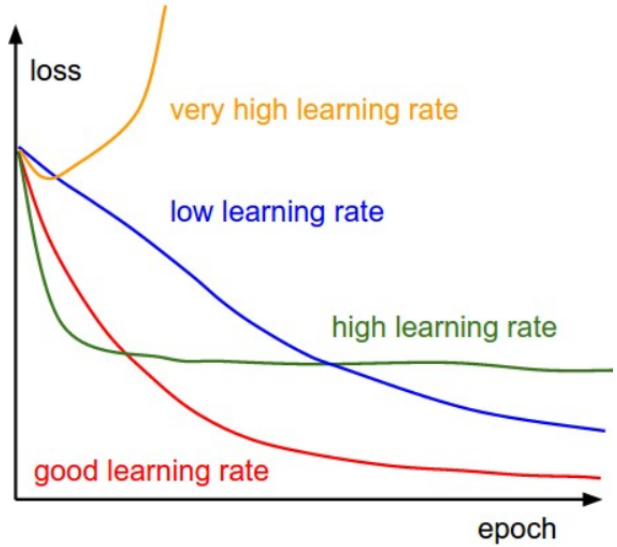
- Let $L(\theta)$ be a loss function or **criterion** (e.g. -log-likelihood, classification error...)
- Aim: learn the model parameters θ that **optimize** the criterion (in general, minimize a loss).
- In machine learning, optimization is usually done using first-order information, i.e. gradient (derivative)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

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**.

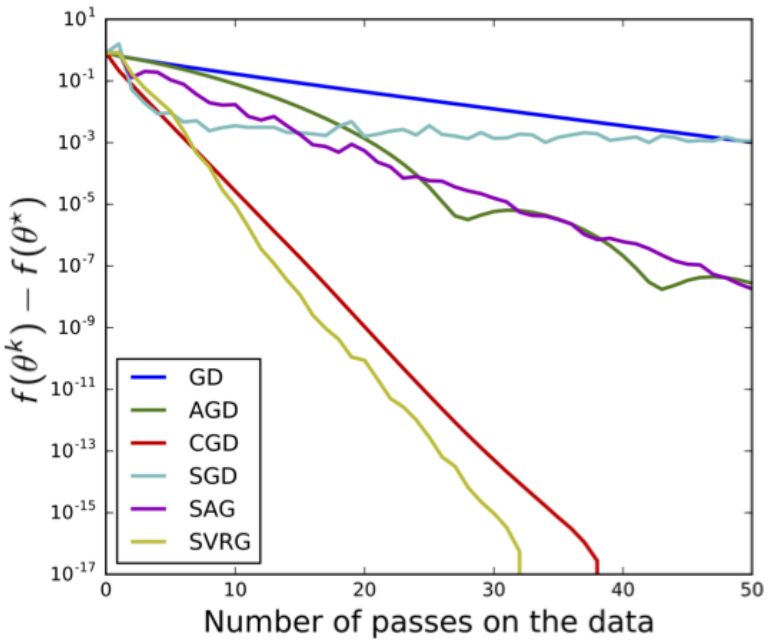


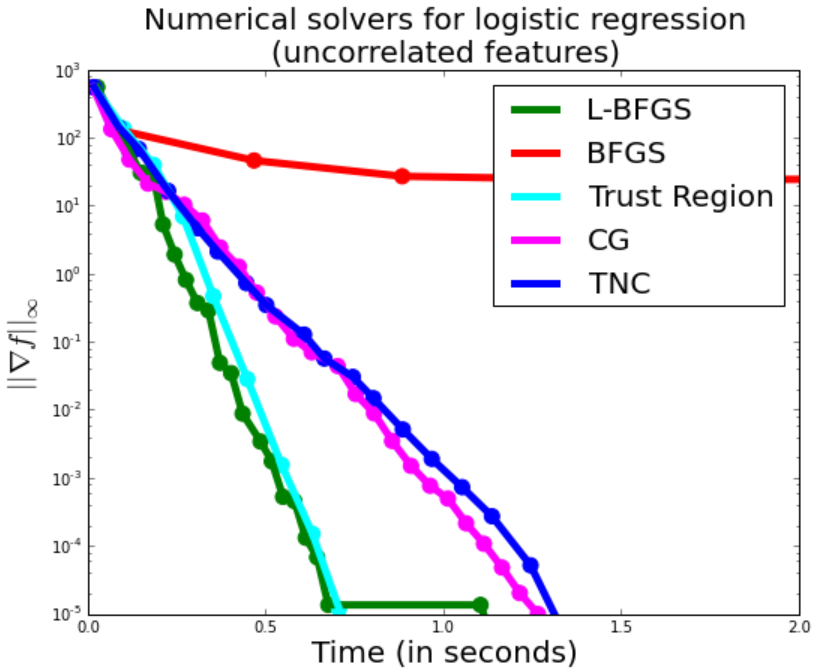
- if λ is very small, longer time to converge, computationally expensive;
- if λ is large, gradient descent may fail to converge and overshoot the minimum.

Optimization of a loss II

Optimization of a loss III

Many solvers out there!

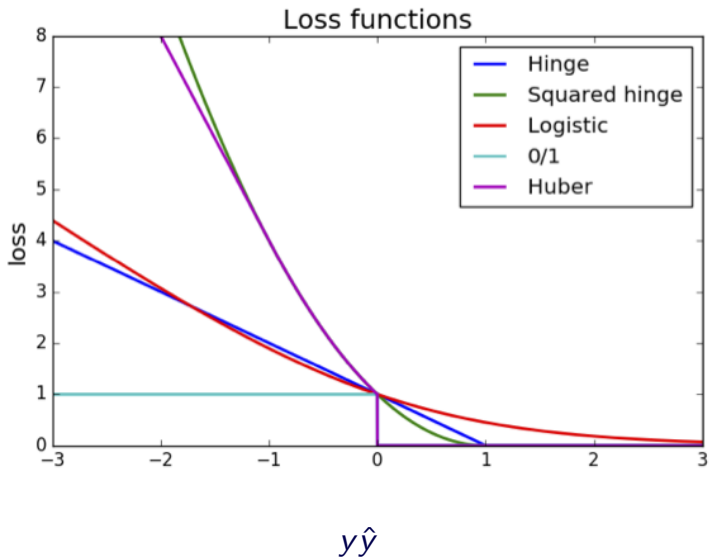




Other classical loss functions for binary classification

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 $\ell(y, \hat{y}) = -4y\hat{y}1_{y\hat{y}<-1} + (1 - y\hat{y})_+^2 1_{y\hat{y}\geq -1}$



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

Outline of day 1

1. Introduction to this course
2. Binary classification
3. Logistic regression
4. Training of the classifier
5. Performance metrics
6. Overfitting, penalization and cross validation

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 $\mathrm{TP} = \sum_{i=1}^{n} 1_{y_i=1,\hat{y}_i=1}$

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

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

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

Confusion matrix

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

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

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

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

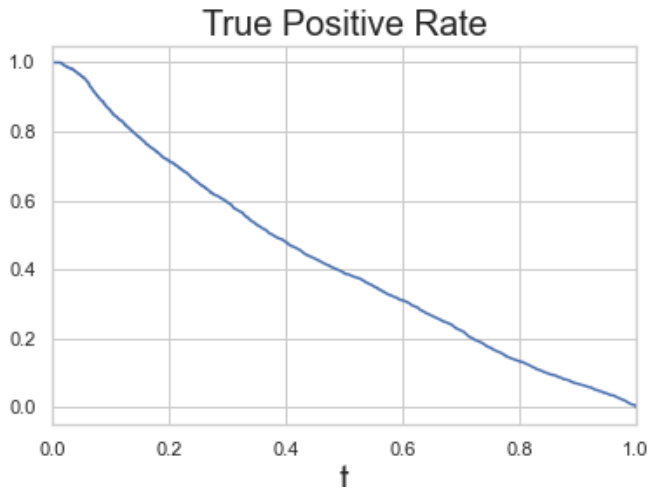
True and false positive rate

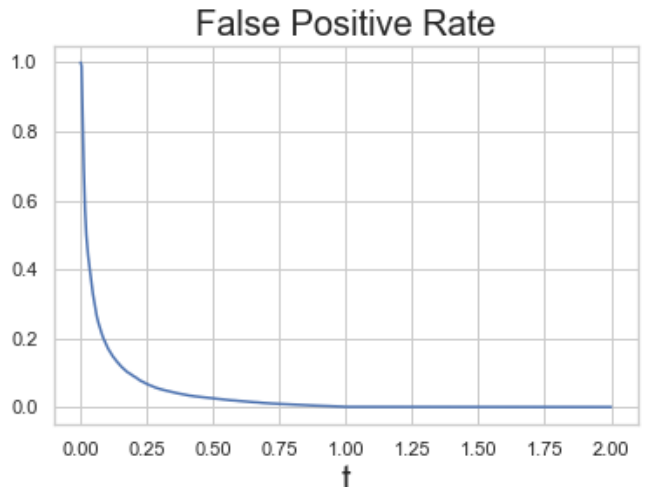
- True positive rate (= recall):

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

- False positive rate (= 1 - specificity):

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





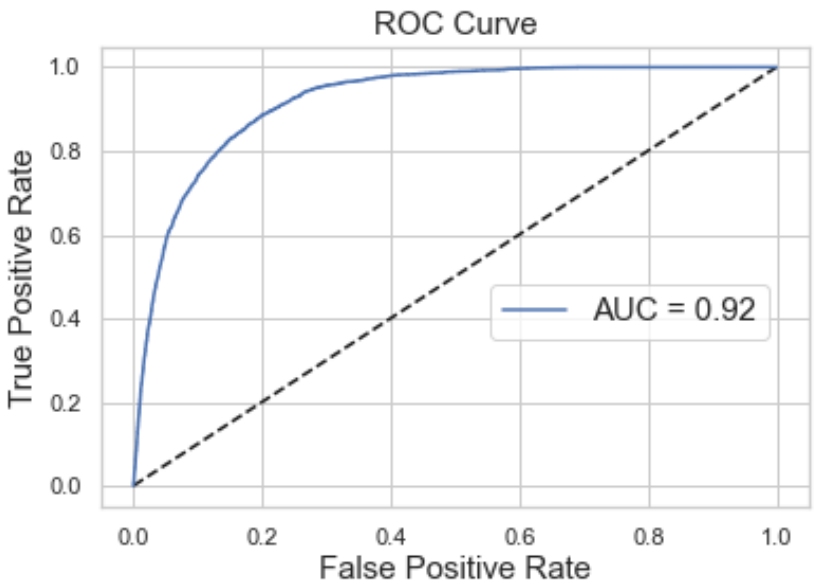
ROC curve

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

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

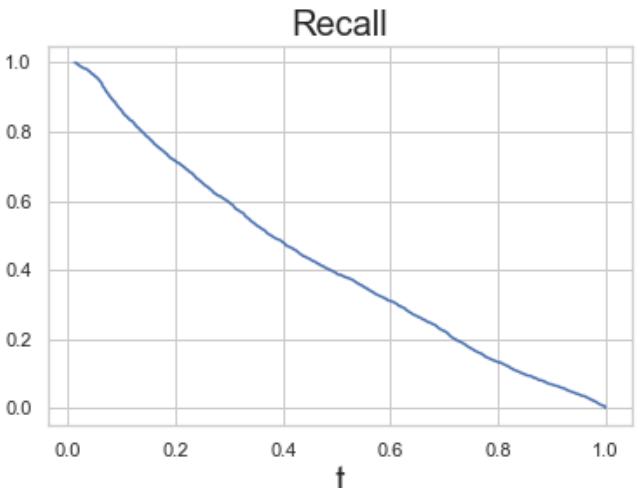
- The **AUC-ROC score** is the Area Under the ROC curve
- The larger the AUC, the better
- A classifier whose ROC curve is uniformly larger than the ROC curve of another classifier is preferable

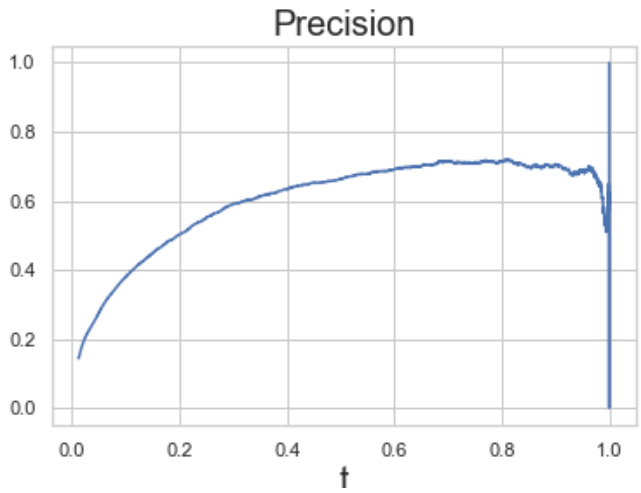
- The line $y = x$ represents the ROC curve of the family of **random classifiers** that output $\hat{y}_i = 1$ at random with proba $t \in [0, 1]$



Precision-recall curve I

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



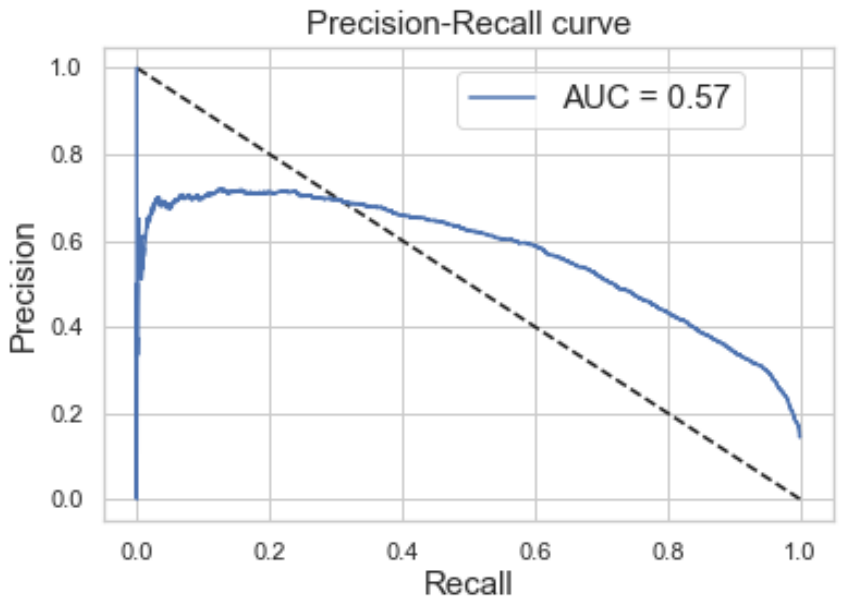


Precision-recall curve II

- 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



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

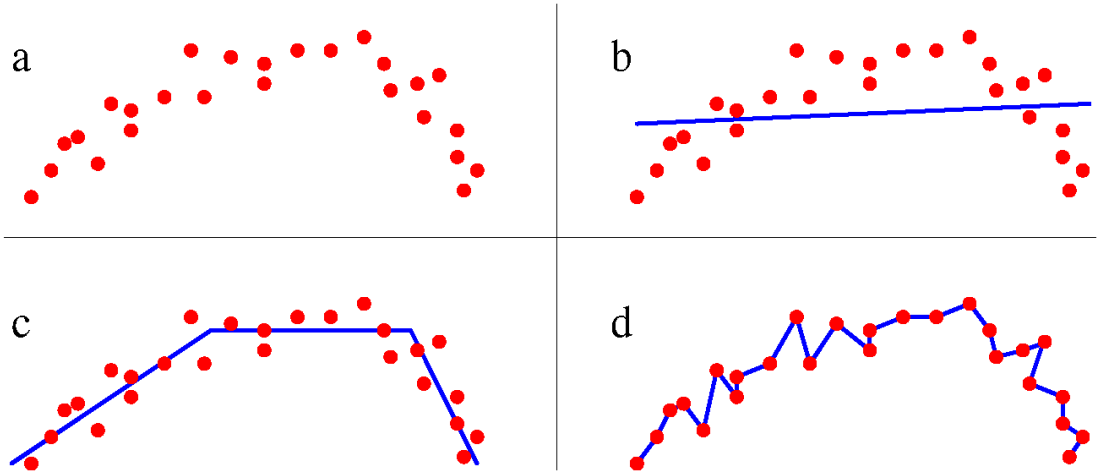
- Split data $\mathcal{D} = \{(y_i, x_i), i = 1, \ldots, n\}$ into a training set and a test set: $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}} = \mathcal{D}$
  - Use $\mathcal{D}_{\text{train}}$ to learn the classifier
  - Use $\mathcal{D}_{\text{test}}$ to evaluate error metrics and performance 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

1. Introduction to this course
2. Binary classification
3. Logistic regression
4. Training of the classifier
5. Performance metrics
6. Overfitting, penalization and cross validation

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**

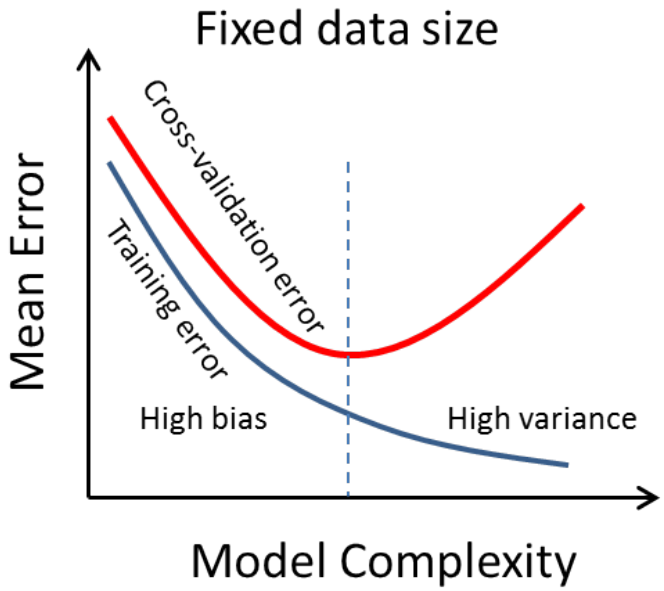


Problem of overfitting

- 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

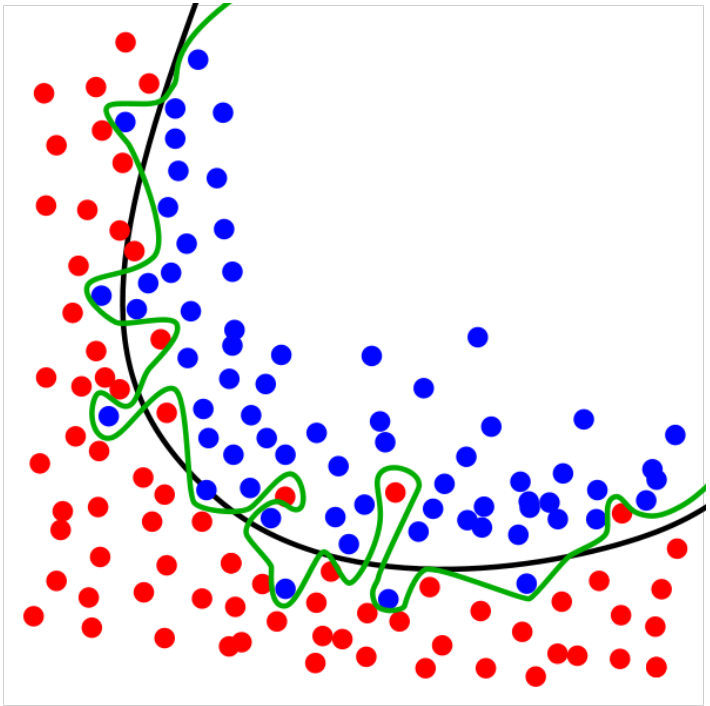
Model selection problem

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



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



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

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

Penalized logisitic regression

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

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

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.

Combinations

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

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

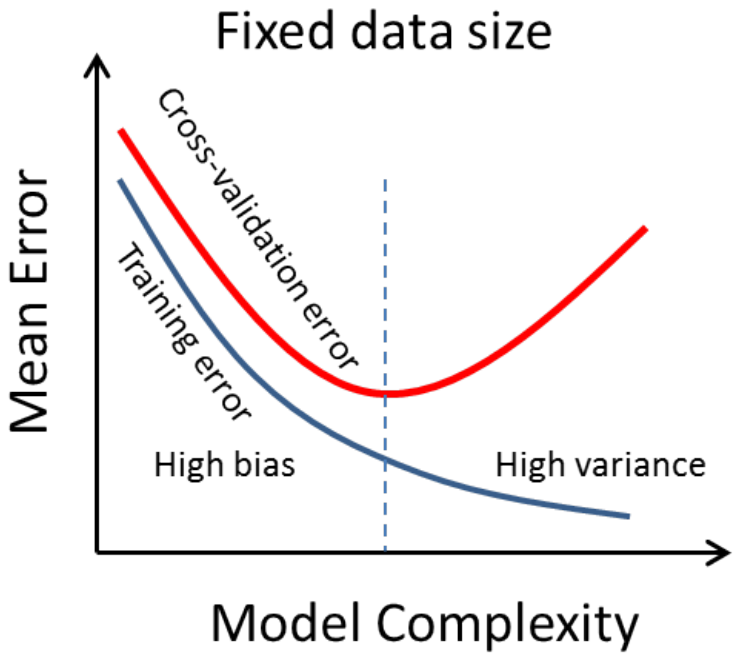
- It has the advantages of both ridge- and ℓ_1-penalization
- $\alpha \geq 0$ balances the two penalizations

Minimization of the penalized loss

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

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

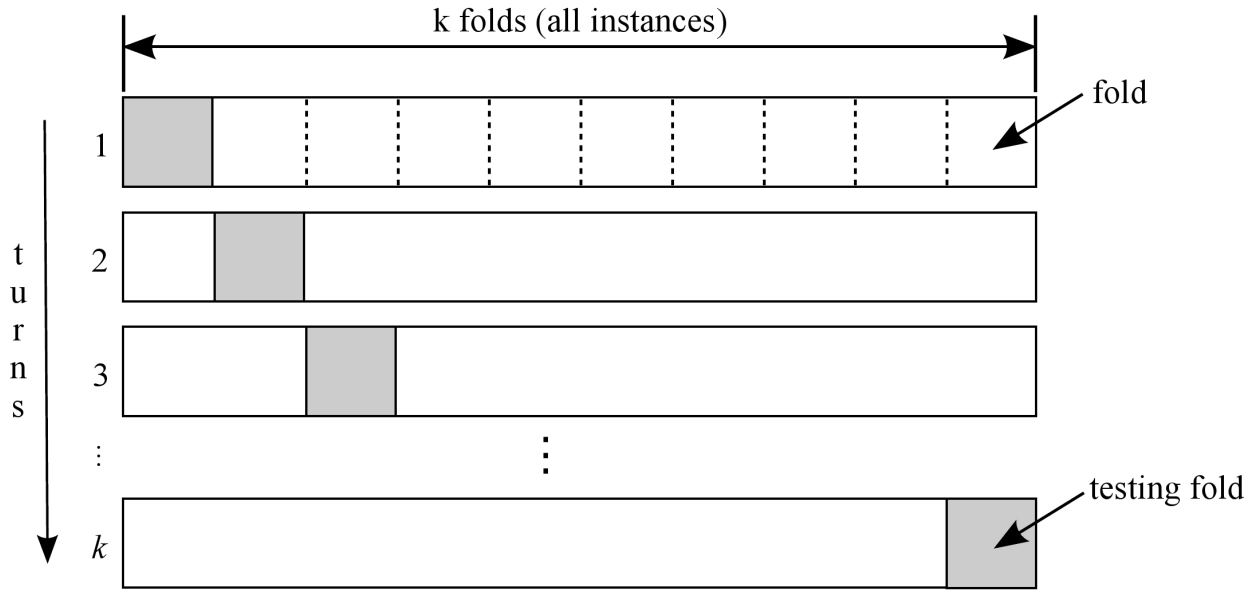
Cross validation

- 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 $K = 10$



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 IV

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

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

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

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

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

This afternoon

- We will put all this into practice

Thank you!