DD2437 – Artificial Neural Networks and Deep Architectures (annda)

Lecture 4: **Practical aspects of ANN approaches to pattern recognition problems**

Pawel Herman

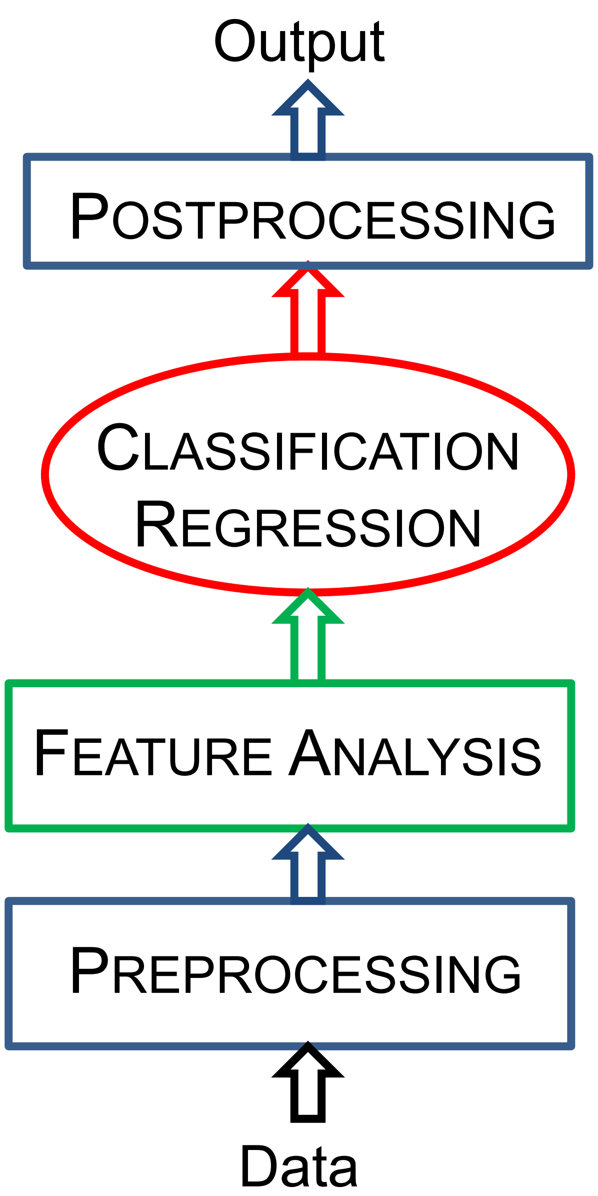
Computational Science and Technology (CST)

KTH Royal Institute of Technology

Lecture overview

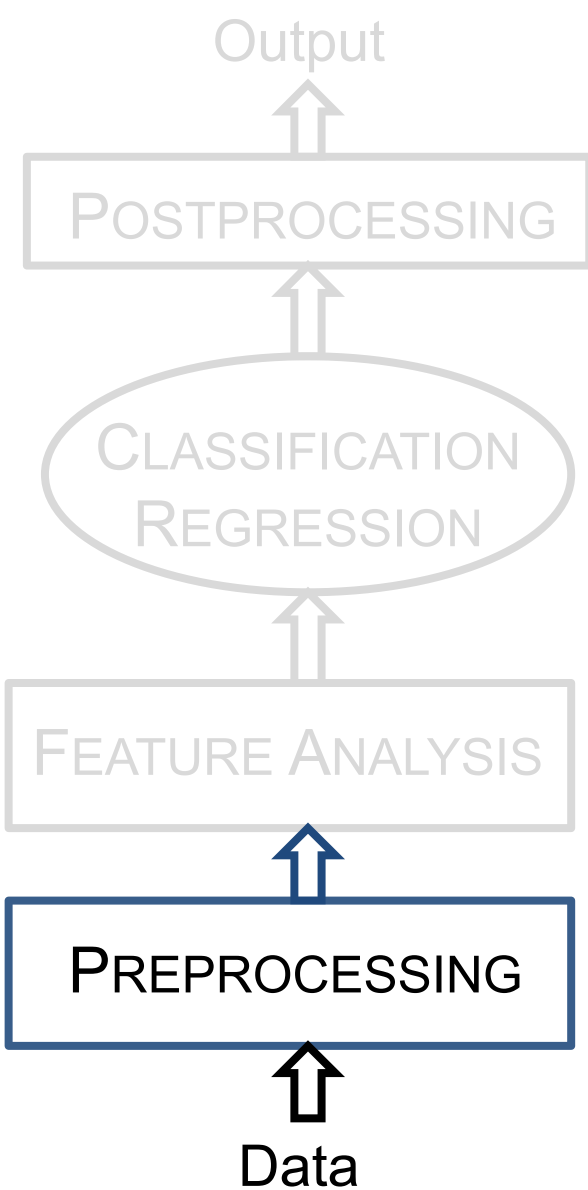
- Data preprocessing and feature extraction
- Error measures
- Parameter optimisation

Pattern recognition pipeline



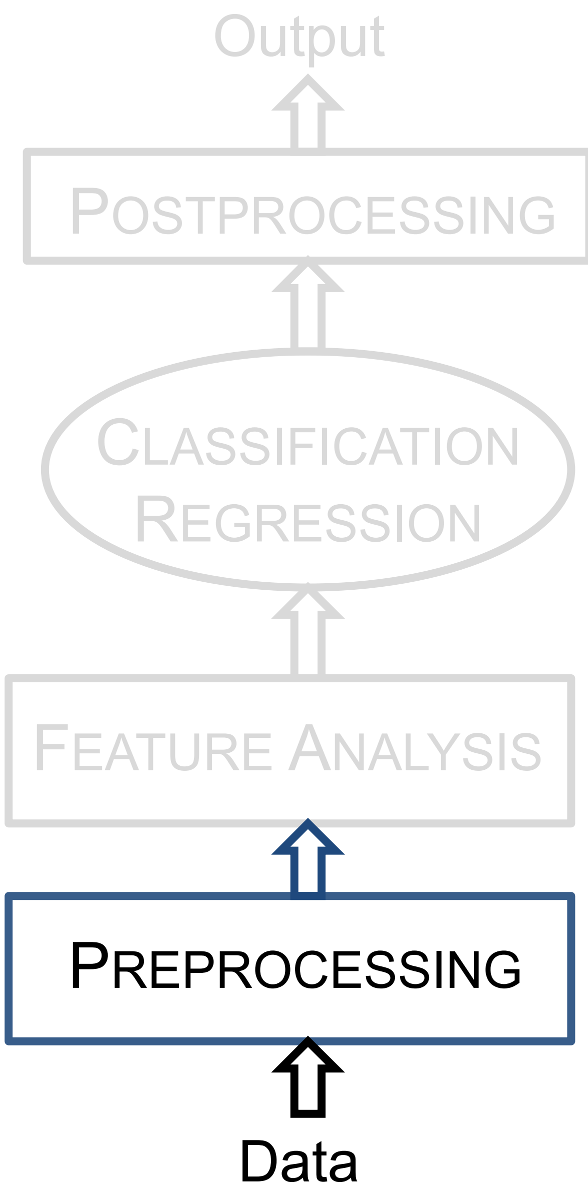
1. Preprocessing
2. Features, low-level data representation
3. Classification / regression with ANN
4. Postprocessing (alternative)

Pattern recognition pipeline



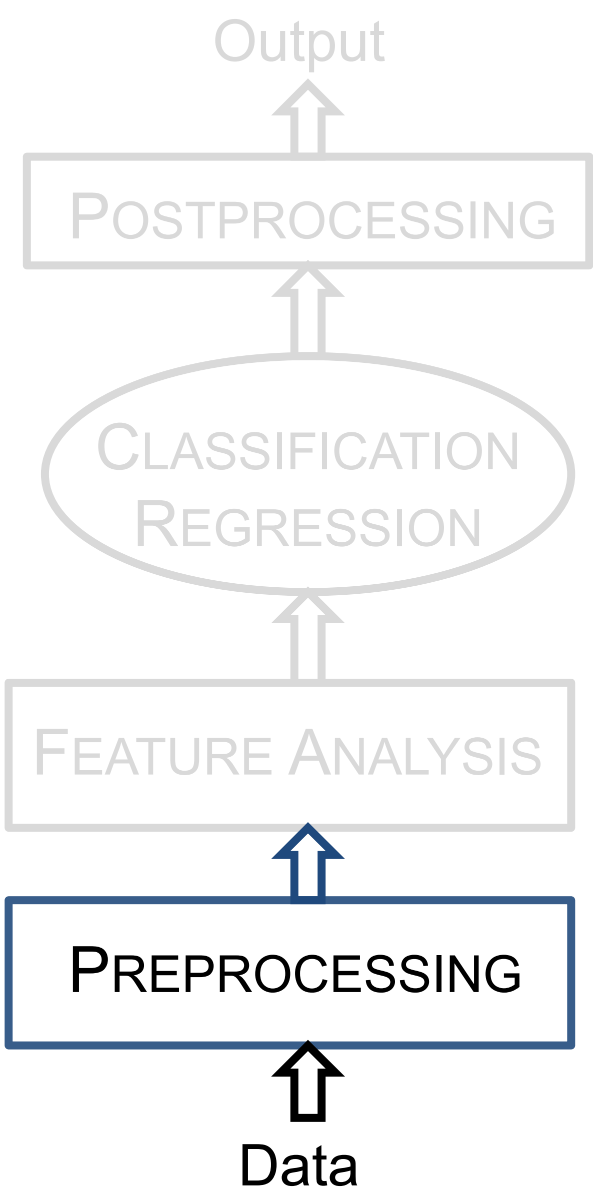
1. Preprocessing
   - familiarise yourself with data and problem
     - what is the objective and assumptions?
     - what data are available?
     - how are/were data generated?
     - type of attributes, their distribution
     - plot data, estimate basic statistics, correlations
     - what is prior knowledge?
   - data quality assessment
   - de-noising, outlier analysis
   - data transformations, normalisation
   - missing data

Pattern recognition pipeline



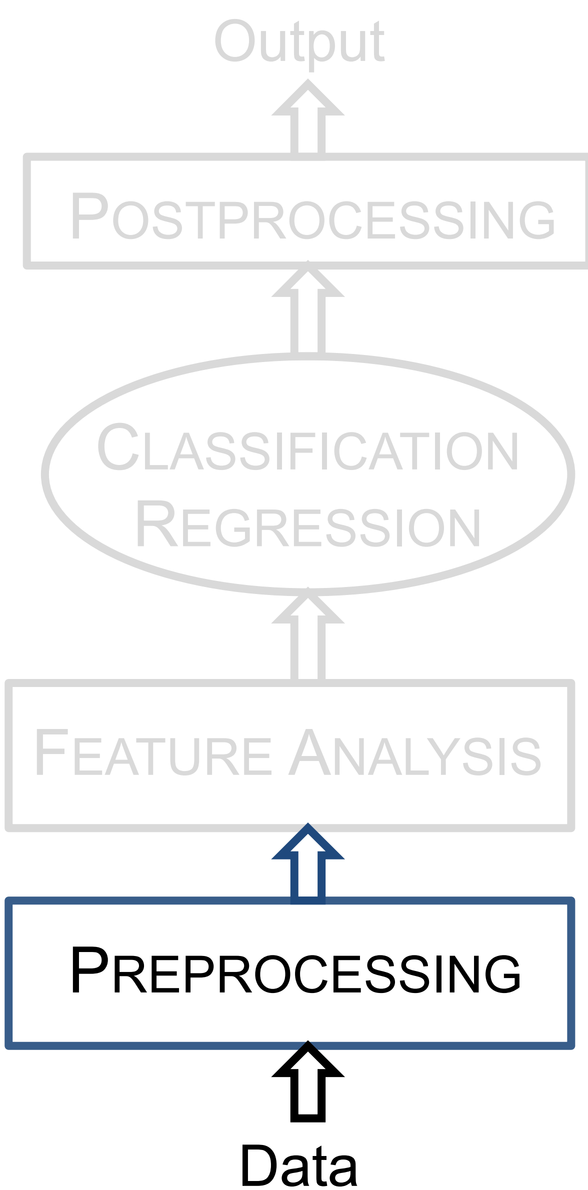
1. Preprocessing
   - familiarise yourself with data and problem
   - data quality assessment
     - train & test data from the same distribution?
     - dimensionality, amount of data
     - dealing with discontinuities
   - de-noising, outlier analysis
   - data transformations, normalisation
   - missing data
   - data augmentation, unbalanced data

Pattern recognition pipeline



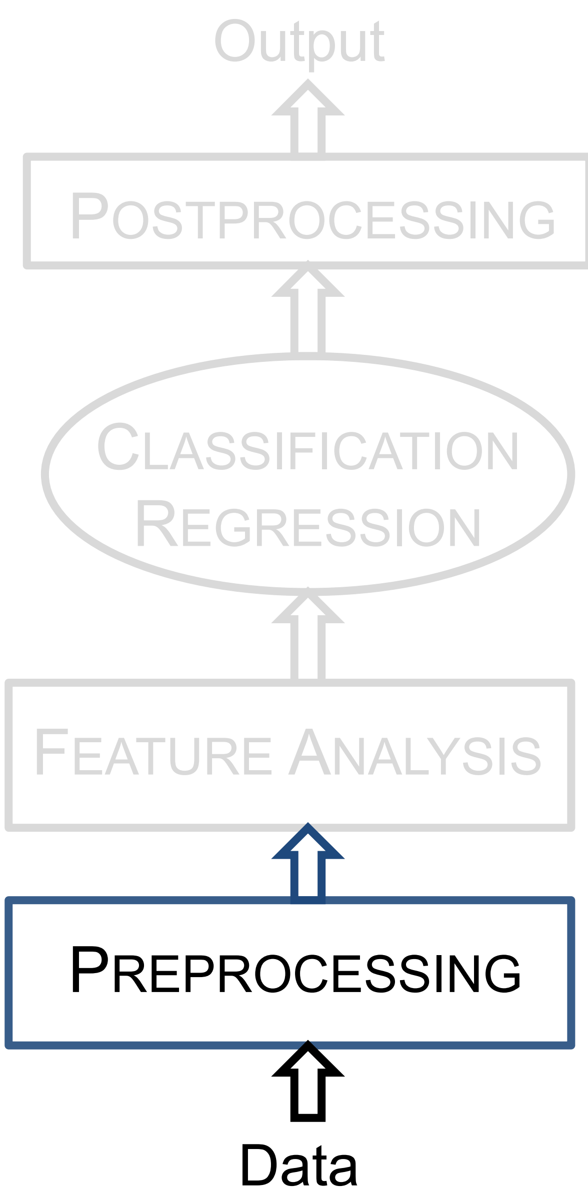
1. Preprocessing
   - familiarise yourself with data and problem
   - data quality assessment
   - de-noising, outlier analysis
     - collect information about noise
     - noise removal
     - outlier detection – remove?
     - filtering
   - data transformations, normalisation
   - missing data
   - data augmentation, unbalanced data

Pattern recognition pipeline



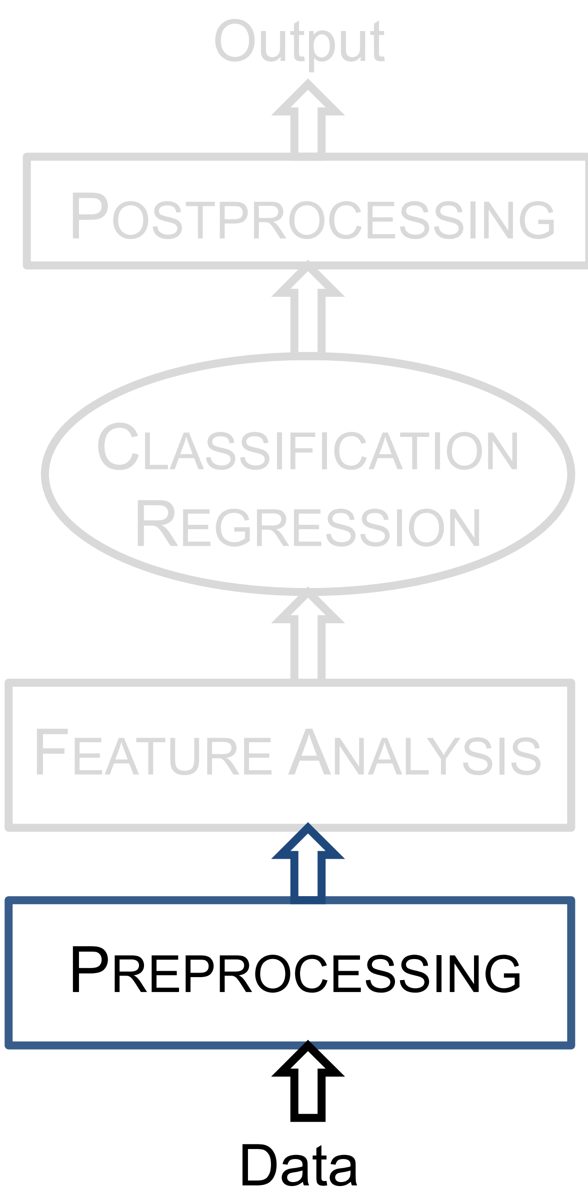
1. Preprocessing
   - familiarise yourself with data and problem
   - data quality assessment
   - de-noising, outlier analysis
   - data transformations, normalisation
     - attribute normalisation
     - whitening
     - scaling (linear, nonlinear, e.g. log)
   - missing data
   - data augmentation, unbalanced data

Pattern recognition pipeline



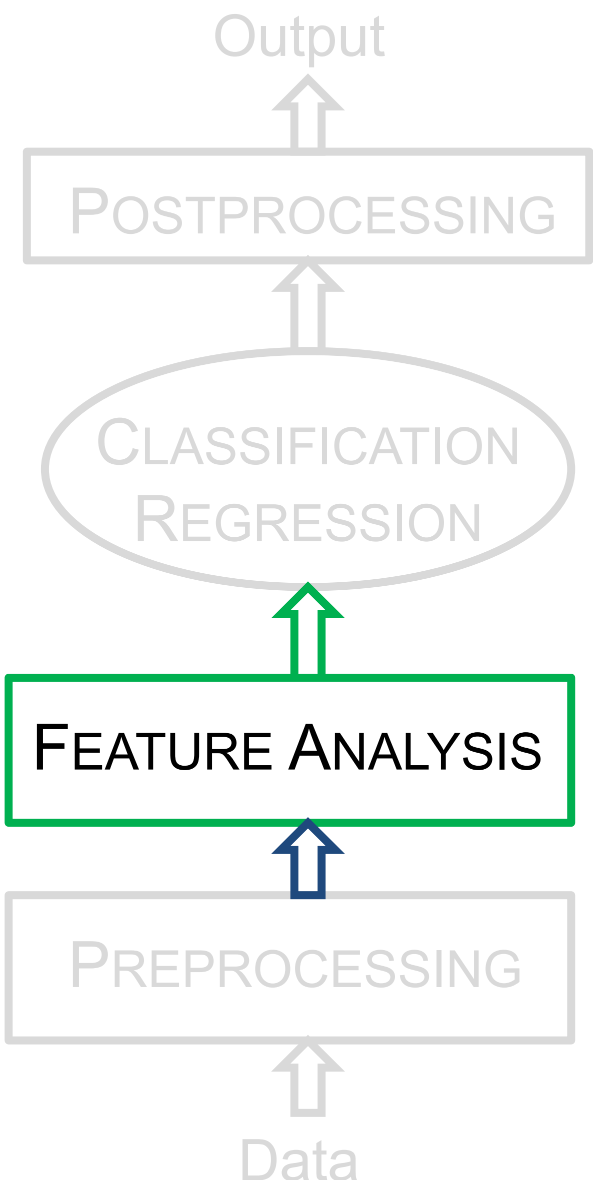
1. Preprocessing
   - familiarise yourself with data and problem
   - data quality assessment
   - de-noising, outlier analysis
   - data transformations, normalisation
   - missing data
     - remove
     - replace with the mean
     - estimate by regression
     - handle by the pattern recognition algorithm
   - data augmentation, unbalanced data

Pattern recognition pipeline



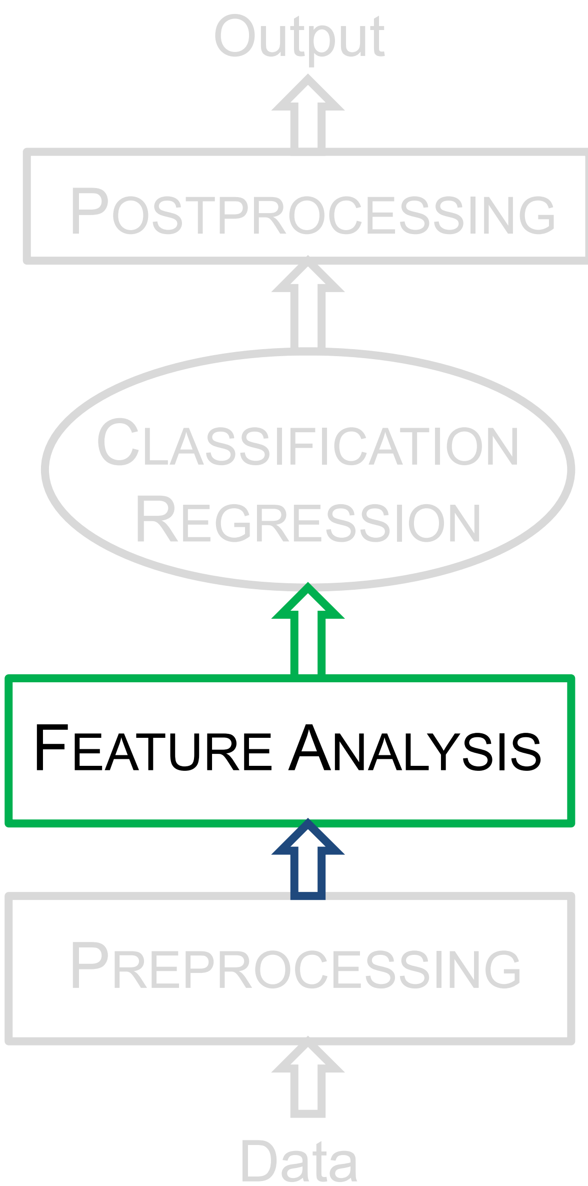
1. Preprocessing
   - familiarise yourself with data and problem
   - data quality assessment
   - de-noising, outlier analysis
   - data transformations, normalisation
   - missing data
   - data augmentation, unbalanced data
     - upsampling, downsampling
     - perturbations introduced to boost generalisation

Pattern recognition pipeline



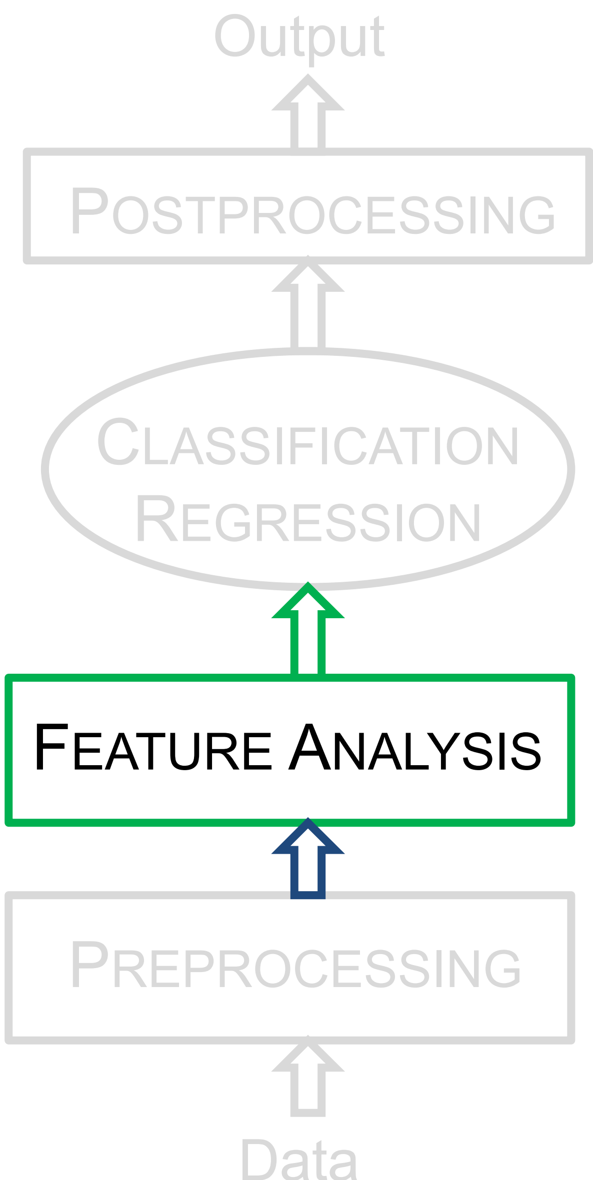
1. Preprocessing
2. Features, low-level data representation
   - dimensionality reduction
     - PCA, SOM, ICA to study data in lower-dim spaces or extract features (projections)
     - decorrelation
   - transformation to a new space
   - feature selection

Pattern recognition pipeline



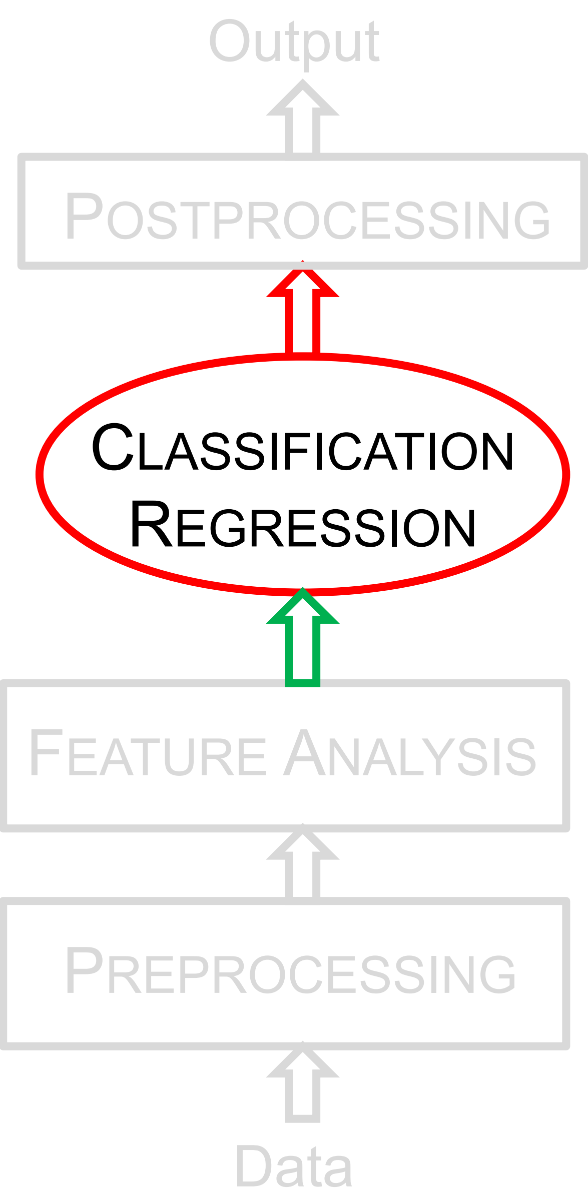
1. Preprocessing
2. Features, low-level data representation
   - dimensionality reduction
   - transformation to a new space
     - low-level data representations, extracting domain specific features
     - invariances (translational, rotational, etc.), symmetries
     - sparsification,redundancy, orthogonalisation
     - encoding, e.g. interval coding
   - feature selection

Pattern recognition pipeline



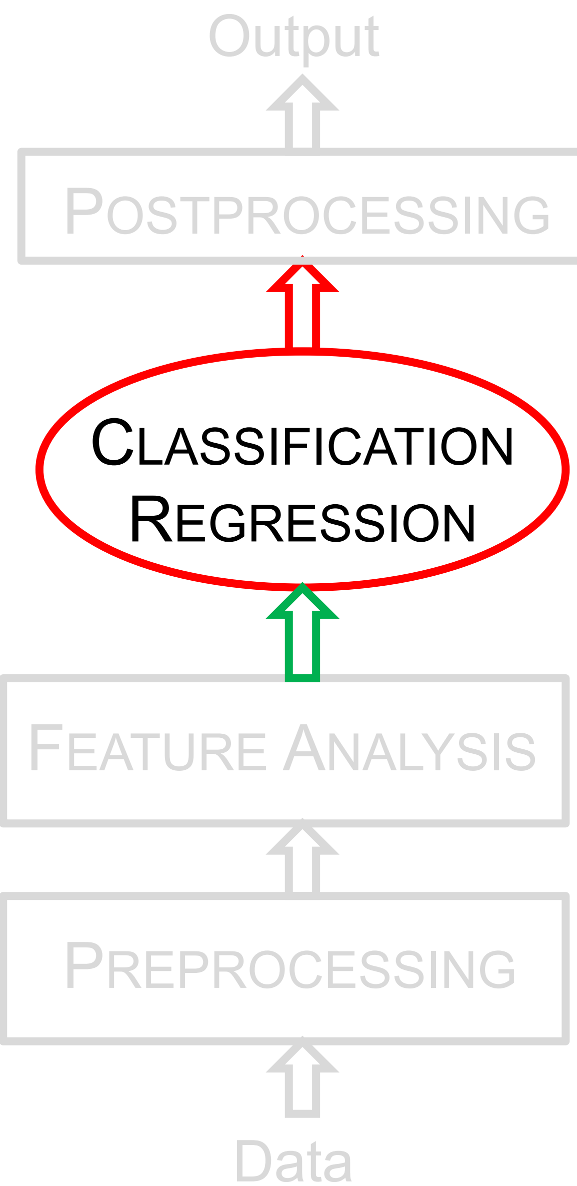
1. Preprocessing
2. Features, low-level data representation
   - dimensionality reduction
   - transformation to a new space
   - feature selection
     - search techniques
     - criteria of evaluation, e.g. filtering, wrapping

Pattern recognition pipeline



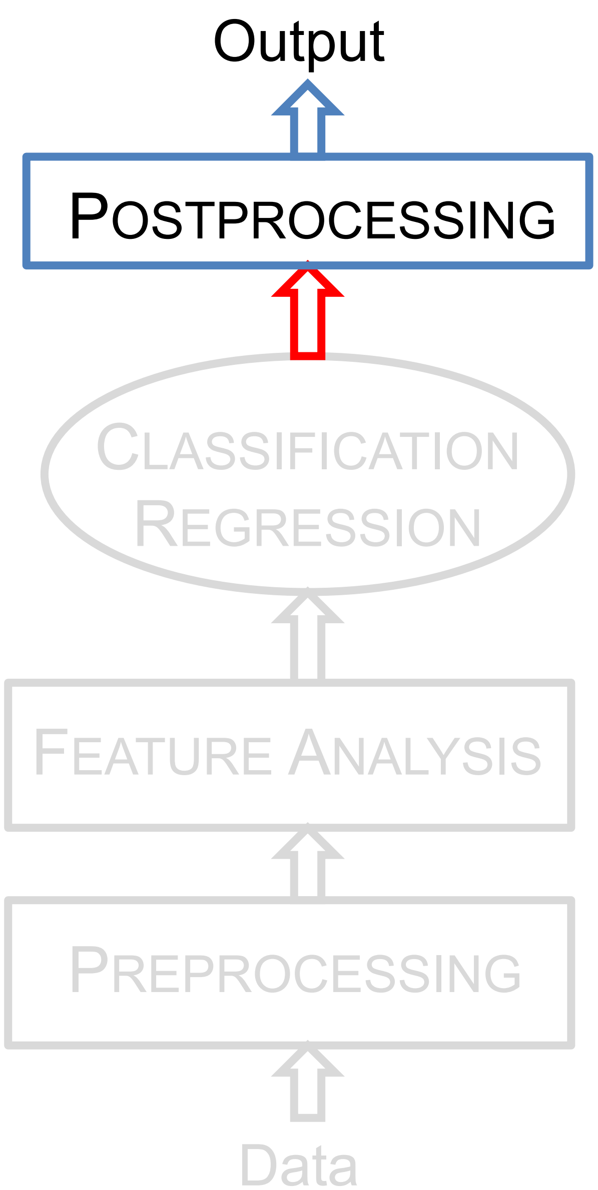
1. Preprocessing
2. Features, low-level data representation
3. Classification / regression with ANN
   - generalisation issues
     - underfitting vs overfitting
     - regularisation, cross-validation
     - assumption about smooth data distribution
   - model selection

Pattern recognition pipeline



1. Preprocessing
2. Features, low-level data representation
3. Classification / regression with ANN
   - generalisation issues
   - problem re-formulation
   - model selection
     - validation, comparison – statistical evidence
     - configuration, hyperparameter optimisation

Pattern recognition pipeline



1. Preprocessing
2. Features, low-level data representation
3. Classification / regression with ANN
4. Postprocessing (alternative)
   - interpretation, visualisation
   - in relation to preprocessing, re-mapping
   - domain-, problem-dependent processing

Training ANNs as an optimisation task

- The notion of error/loss function (objective function)
  - loss function should correspond to the framing of the specific modeling problem – regression, classification
  - choice of the loss function is related to the configuration of the output layer (*activation function*, the number of outputs) – framing of the problem
  - loss function and learning algorithm

Training ANNs as an optimisation task

- The notion of error/loss function (objective function)
  - loss function should correspond to the framing of the specific modeling problem – regression, classification
  - choice of the loss function is related to the configuration of the output layer (*activation function*, the number of outputs) – framing of the problem
  - loss function and learning algorithm
  - *“the cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared…..it is therefore important that the function faithfully represents our design goals. If we choose a poor error function and obtain unsatisfactory results, the fault is ours for badly specifying the goal of the search”*

Maximum likelihood (ML) framework

- To find the best statistical estimates of parameters from training data:

"Maximum likelihood seeks to find the optimum values for the parameters by maximizing a likelihood function derived from the training data". (Bishop, 1998)

- ML loss function estimates how closely the distribution of predictions made by a model matches the distribution of target variables in the training data
  - *a cross-entropy* between the empirical distribution defined by the training set and the probability distribution defined by model (e.g. targets in classification)
  - In regression, MSE can be seen as the cross-entropy between the distribution of the model predictions and the distribution of the target variable

Maximum likelihood (ML) framework

- To find the best statistical estimates of parameters from training data:

"Maximum likelihood seeks to find the optimum values for the parameters by maximizing a likelihood function derived from the training data". (Bishop, 1998)

- ML loss function estimates how closely the distribution of predictions made by a model matches the distribution of target variables in the training data
  - *a cross-entropy* between the empirical distribution defined by the training set and the probability distribution defined by model (e.g. targets in classification)
  - In regression, MSE can be seen as the cross-entropy between the distribution of the model predictions and the distribution of the target variable
- "Consistency" of the maximum likelihood estimators (the more data we have the closer the empirical distribution can be reproduced)

Common choices under ML framework

- The most popular choices
  - Cross-entropy with sigmoidal activation function in a single (binary classification) or multiple outputs (multi-class)

Binary classification: $$L(y,t)=\sum_{i=1}^{N} t^{(i)}\log y^{(i)}+\left(1-t^{(i)}\right)\log\left(1-y^{(i)}\right)$$

Multi-class (multi-label output): $$L(y,t)=\sum_{i=1}^{N}\sum_{c=1}^{C} t_c^{(i)}\log y_c^{(i)}$$

$$y_c=\frac{\exp(\beta y_c)}{\sum_{j=1}^{C}\exp(\beta y_j)}$$ softmax to approximate probabilities

  - MSE with linear activation function for regression problems

Common choices under ML framework

- The most popular choices
  - Cross-entropy with sigmoidal activation function in a single (binary classification) or multiple outputs (multi-class)
  - MSE with linear activation function for regression problems
- Alternative options
  - Mean Squared Logarithmic Error (MSLE)

$$L(y,t) = \frac{1}{N}\sum_{i=1}^{N}\left(\log(y^{(i)}+1) - \log(t^{(i)}+1)\right)^2$$

  - Mean Absolute Error Loss (MAE)

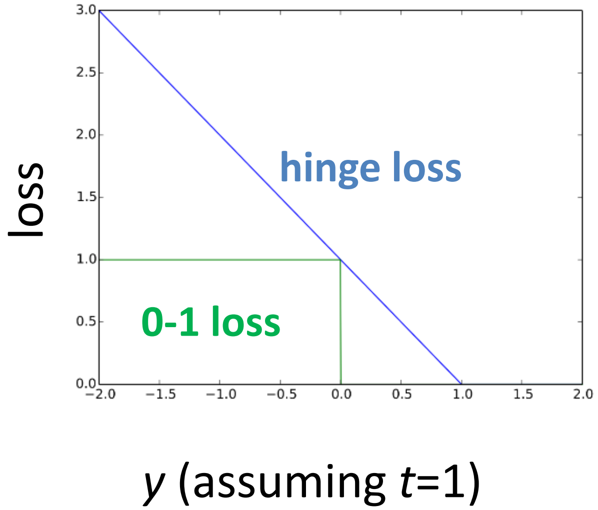
$$L(y,t) = \frac{1}{N}\sum_{i=1}^{N}\left|y^{(i)} - t^{(i)}\right|$$

Common choices under ML framework

- The most popular choices
  - Cross-entropy with sigmoidal activation function in a single (binary classification) or multiple outputs (multi-class)
  - MSE with linear activation function for regression problems

- Alternative options
  - Mean Squared Logarithmic Error (MSLE)
  - Mean Absolute Error Loss (MAE)
  - (Square) Hinge Loss for classification

$$L(y,t) = \sum_{i=1}^{N} \max\left(0,\, 1 - y^{(i)} \cdot t^{(i)}\right)$$

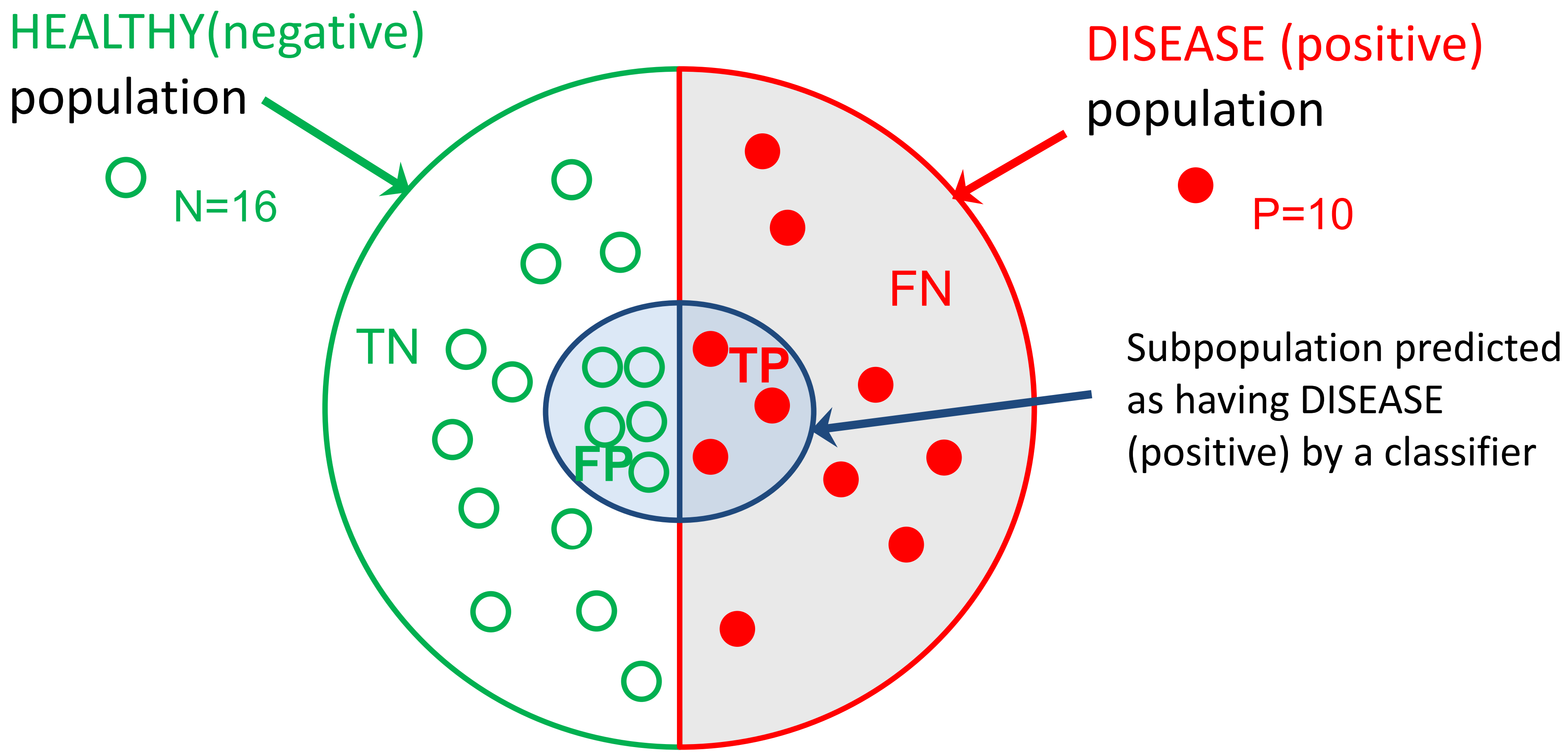


Error/loss function vs performance metrics

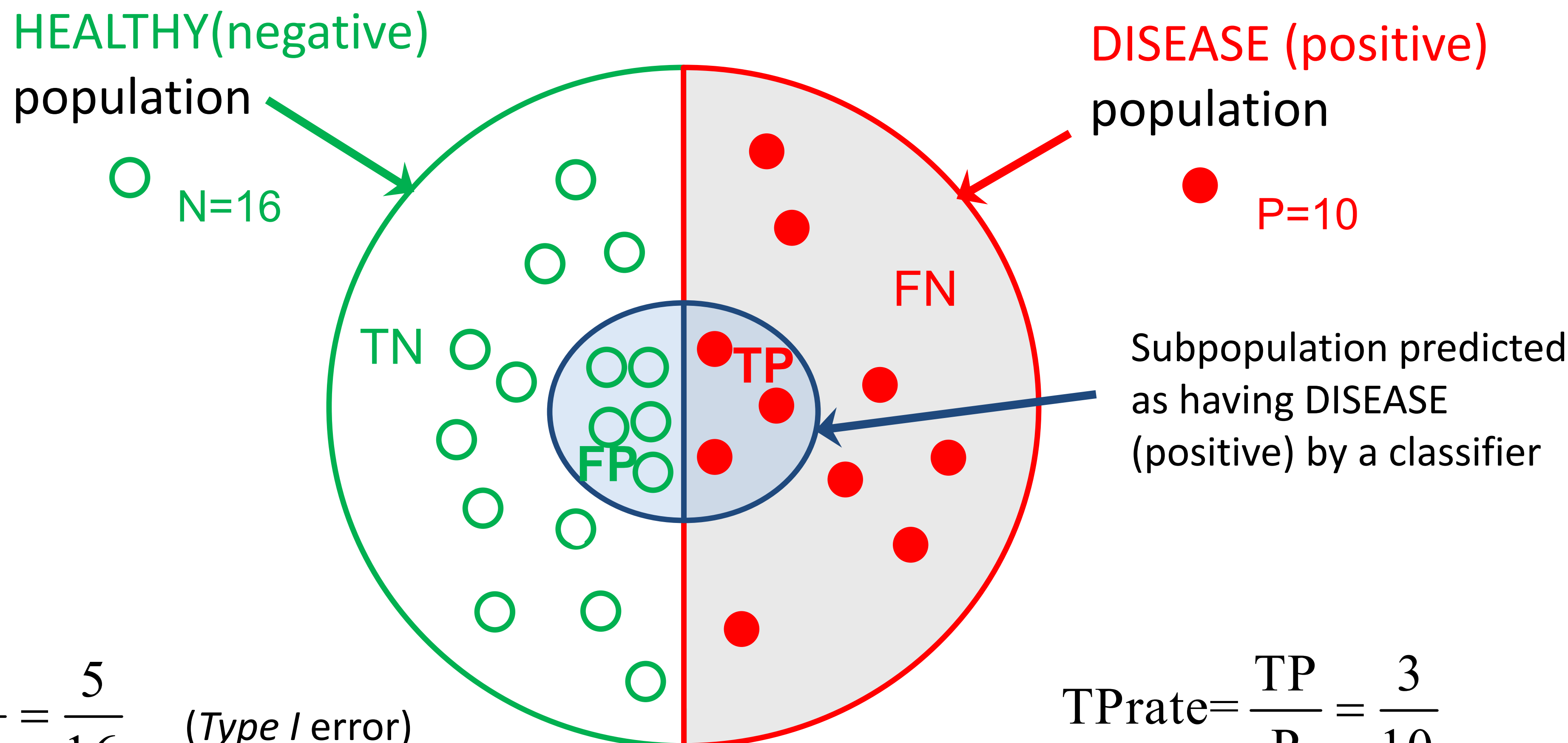
- Error/loss function used for training a neural network does not have to be the same as a problem-dependent performance metric

- In concrete applications, we decide on the measure of performance (related to key performance indicators) and its specific metric
  - Particularly common for classification-type problems
  - e.g. accuracy for classification tasks *BUT does it suffice*?

Specificity vs sensitivity in classification/diagnostics



Specificity vs sensitivity in classification/diagnostics



$$\text{FPrate} = \frac{\text{FP}}{\text{N}} = \frac{5}{16}$$ (*Type I* error)

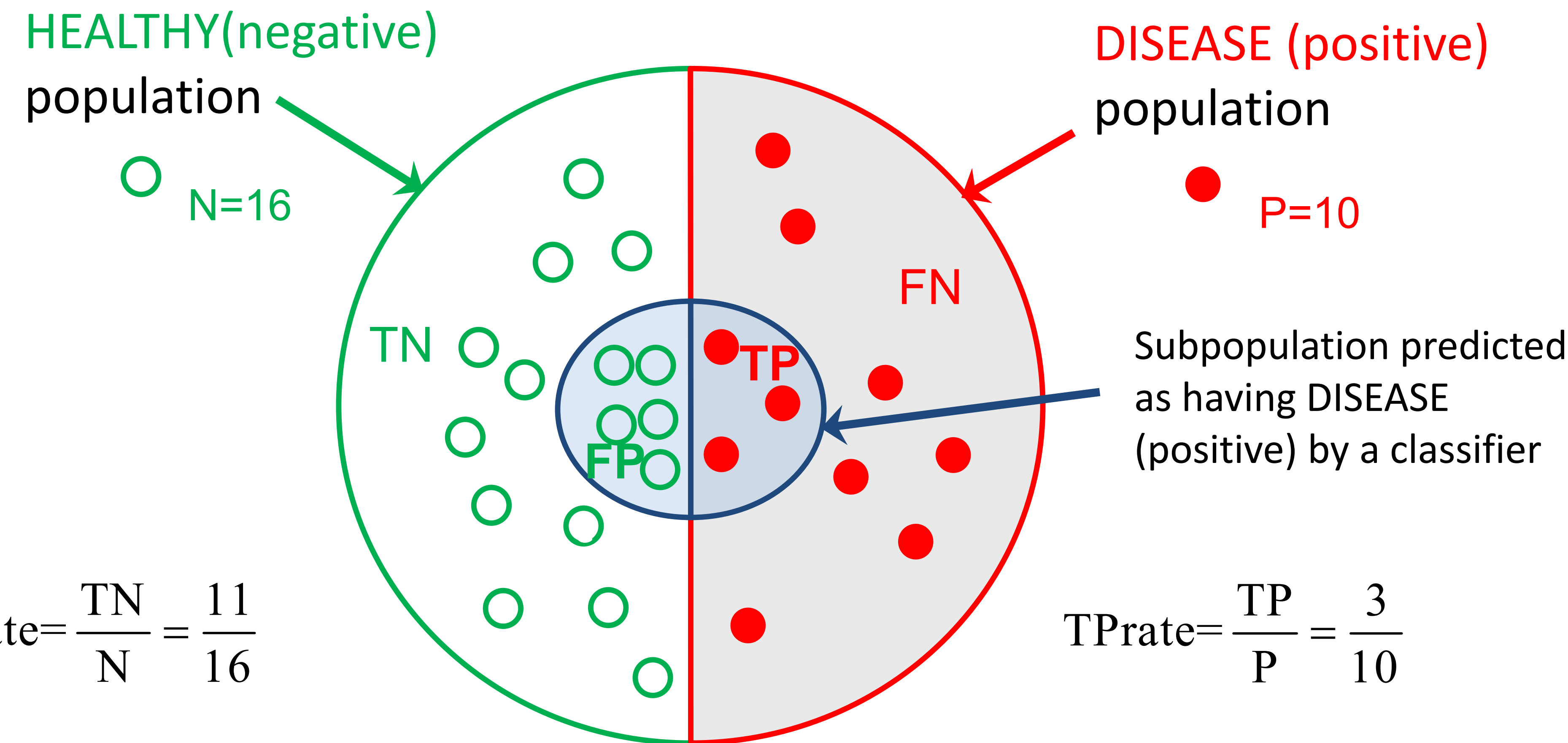
$$\text{TPrate} = \frac{\text{TP}}{\text{P}} = \frac{3}{10}$$

$$\text{TNrate} = \frac{\text{TN}}{\text{N}} = 1 - \text{FPrate} = \frac{11}{16}$$

$$\text{FNrate} = \frac{\text{FN}}{\text{P}} = 1 - \text{TPrate} = \frac{7}{10}$$

(*Type II* error)

Specificity vs sensitivity in classification/diagnostics



$$\mathrm{TNrate} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{11}{16}$$

$$\mathrm{TPrate} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{3}{10}$$

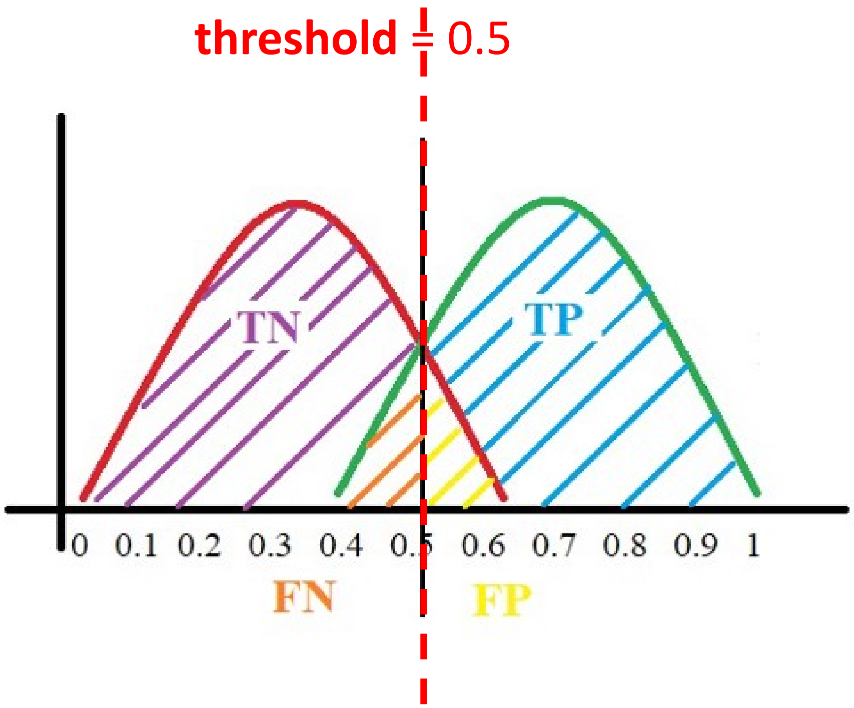
$$\textbf{Specificity}\ (\mathrm{selectivity}) = \mathrm{TNrate} = \frac{\mathrm{TN}}{\mathrm{TN+FP}} = \frac{11}{16}$$

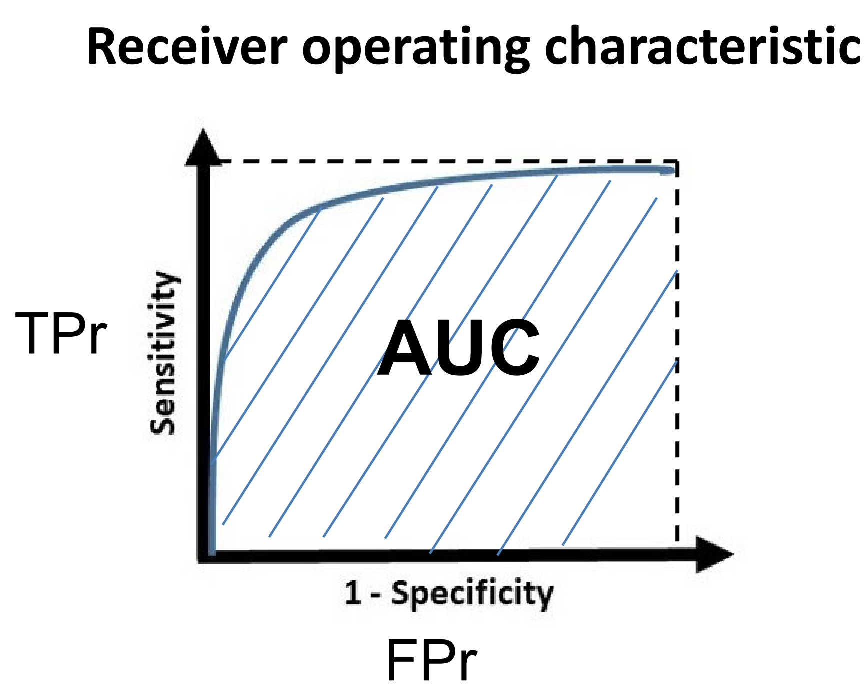
$$\textbf{Precision} = \frac{\mathrm{TP}}{\mathrm{TP+FP}} = \frac{3}{8}$$

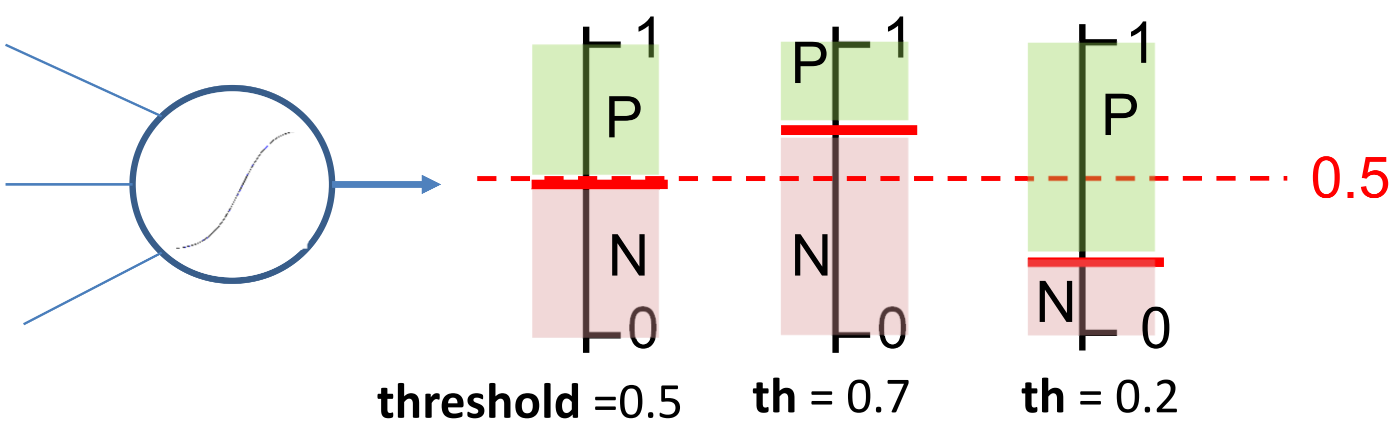
$$\textbf{Sensitivity}\ (\textbf{Recall}) = \mathrm{TPrate} = \frac{\mathrm{TP}}{\mathrm{TP+FN}} = \frac{3}{10}$$

$$\textbf{Fscore} = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

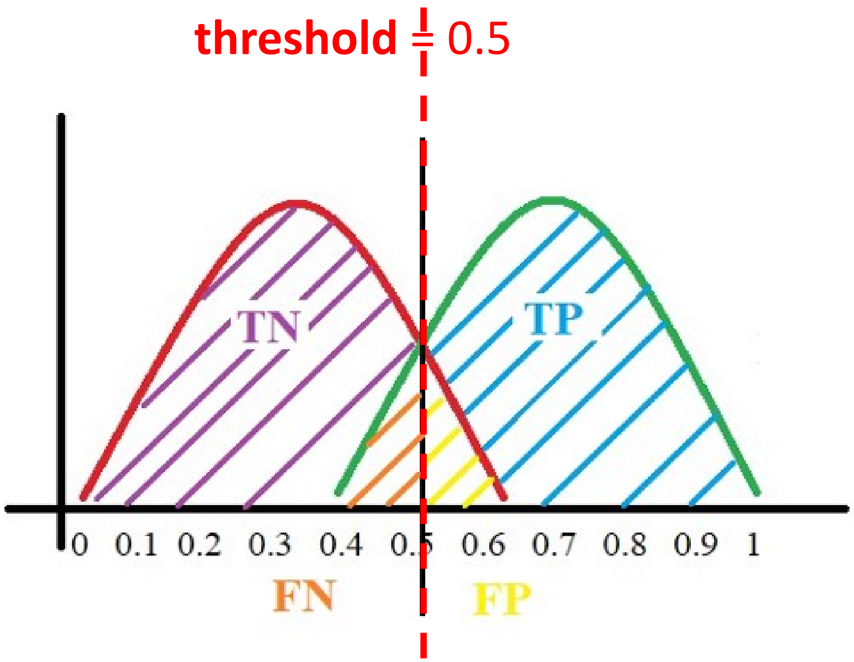
ROC curve in classification/diagnostics



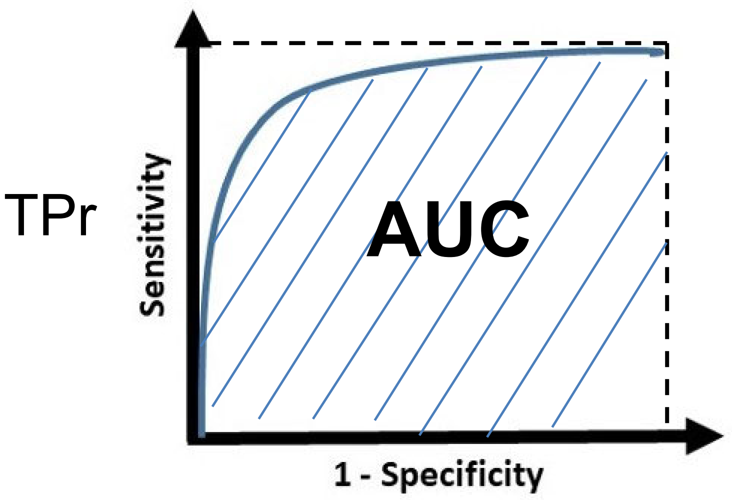


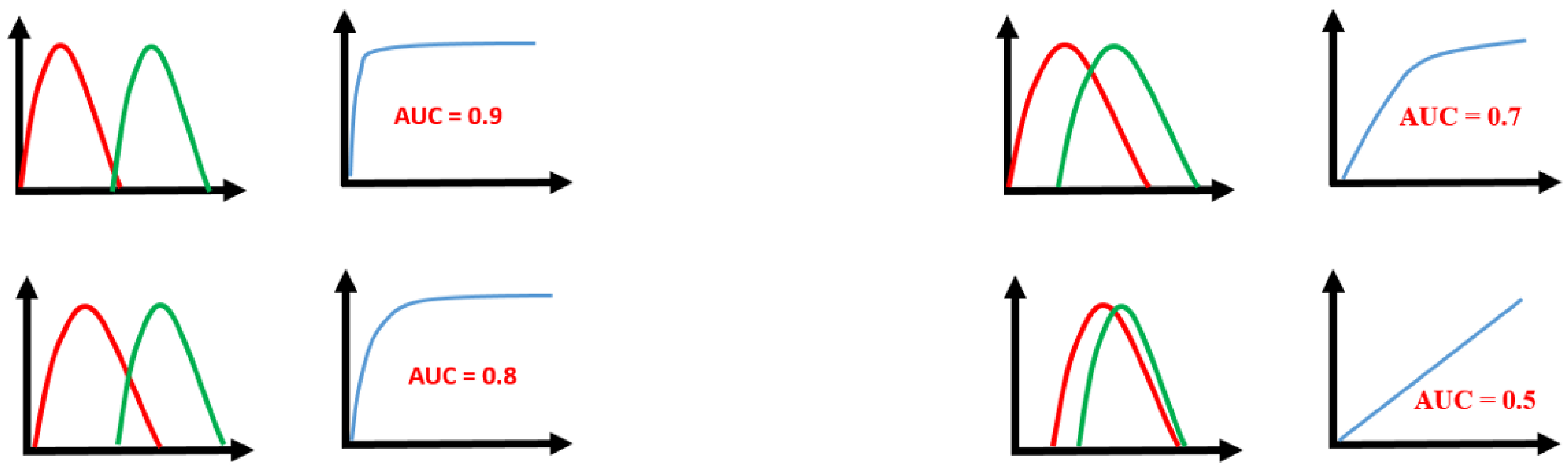


ROC curve in classification/diagnostics



Receiver operating characteristic

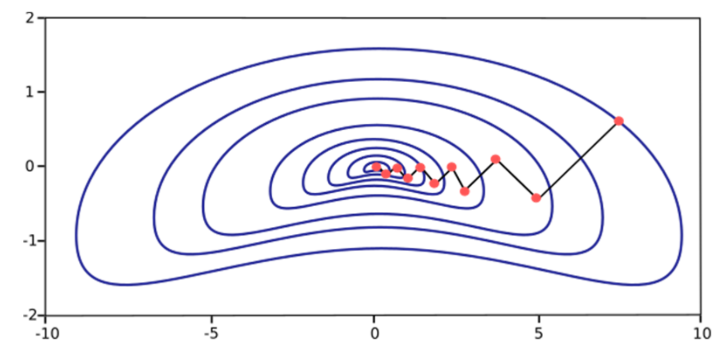




Outline of optimisation algorithms

- Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$

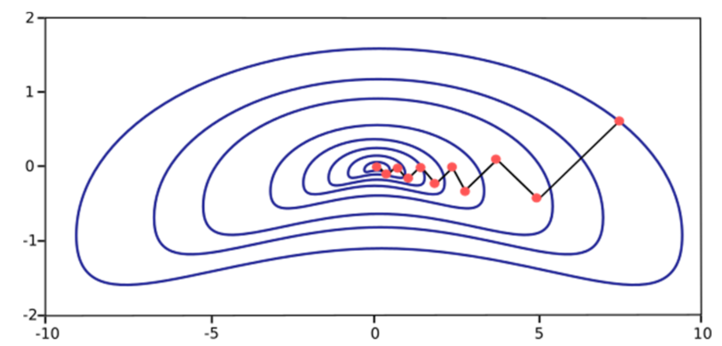


- Gradient descent with extensions
  - momentum, adaptive moment
  - adaptive (even individual) learning rates
  - quickprop (local error surface approximation)
- Iterative optimisation as a line search
- Conjugate gradients (or scaled conjugate gradients)
- Newton's, quasi-Newton and Gauss-Newton approaches
- The Levenberg-Marquardt algorithm

Gradient descent: momentum and mini-batch

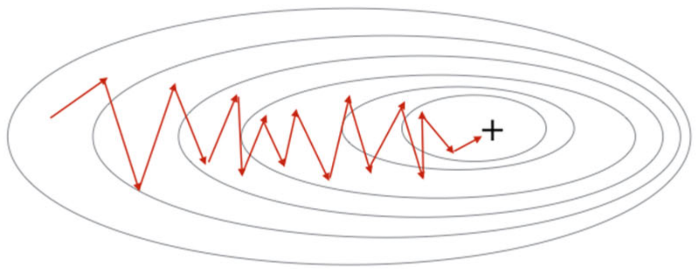
Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$

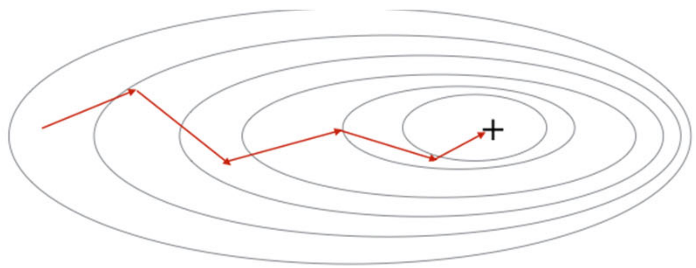


Stochastic gradient descent

SGD



Mini-batch GD



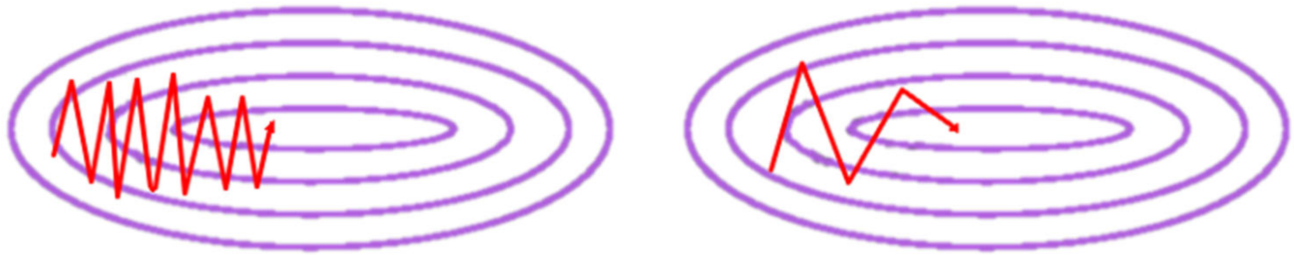
https://datascience-enthusiast.com/DL/Optimization_methods.html

Gradient descent: momentum

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$

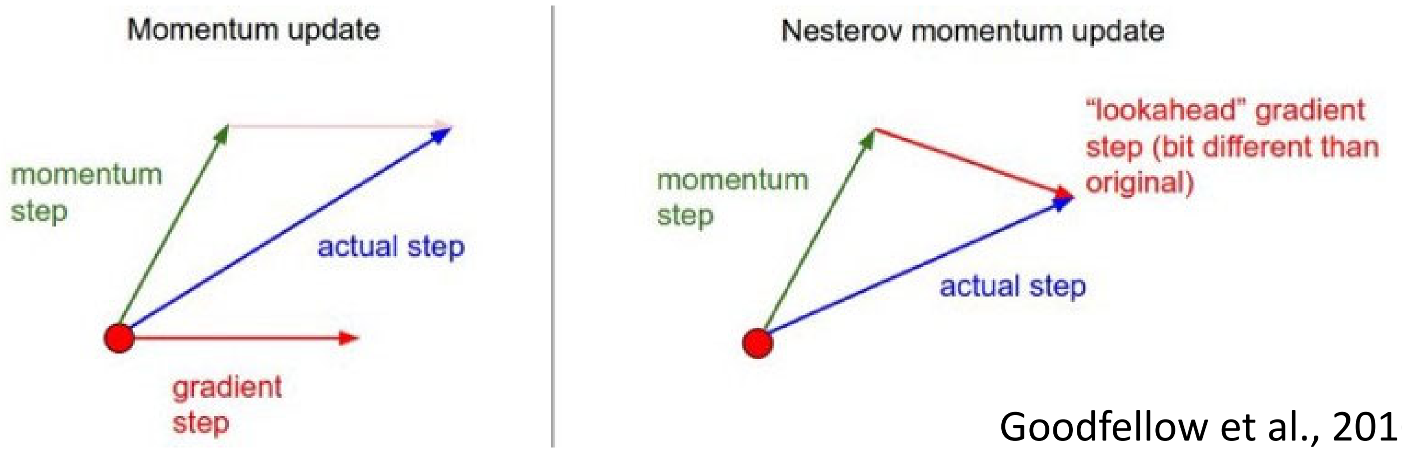
…. and beyond



SGD

SGD with momentum

SGD with Nesterov momentum (Nesterov Accelerated Gradient, NAG)



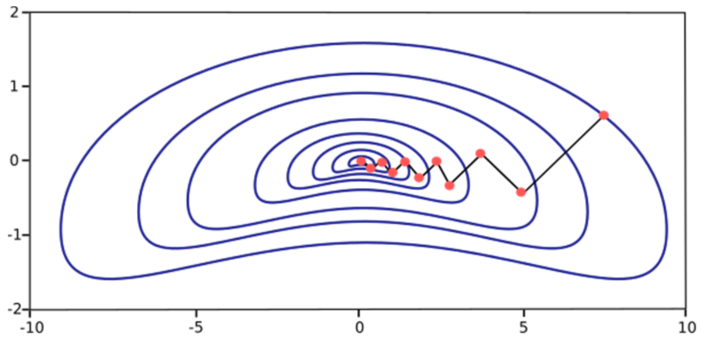
Adapted from Stanford, CS231 class

Goodfellow et al., 2016

Adaptive Gradient Descent – AdaGrad

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$



…. and beyond

Adaptive Gradient Descent (AdaGrad) – adaptive learning rate

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \frac{\mathbf{g}^{(i)}}{\delta + \sqrt{\mathbf{r}^{(i)}}}, \qquad \mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} + \left(\mathbf{g}^{(i)}\right)^2$$

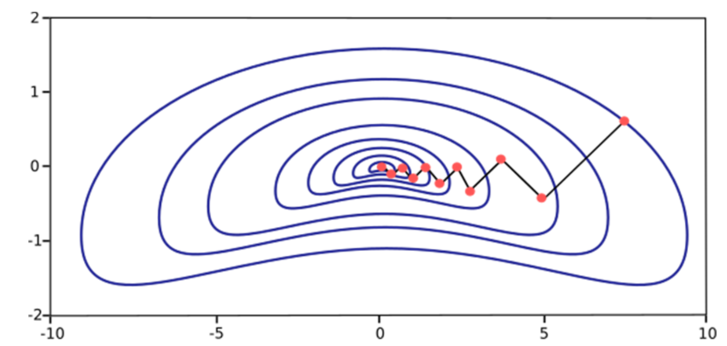
The main problem is quick shrinking of the learning rate -> vanishing gradients

Adapted from https://towardsdatascience.com/

Adaptive Gradient Descent – AdaGrad

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$



.... and beyond

Adaptive Gradient Descent (AdaGrad) – adaptive learning rate

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \frac{\mathbf{g}^{(i)}}{\delta + \sqrt{\mathbf{r}^{(i)}}}, \qquad \mathbf{r}^{(i+1)} = \mathbf{r}^{(i)} + \left(\mathbf{g}^{(i)}\right)^2$$

AdaDelta – AdaGrad with exponentially decaying average (*RMSProp*)

$$\mathbf{r}^{(i+1)} = \rho\mathbf{r}^{(i)} + (1-\rho)\left(\mathbf{g}^{(i)}\right)^2$$

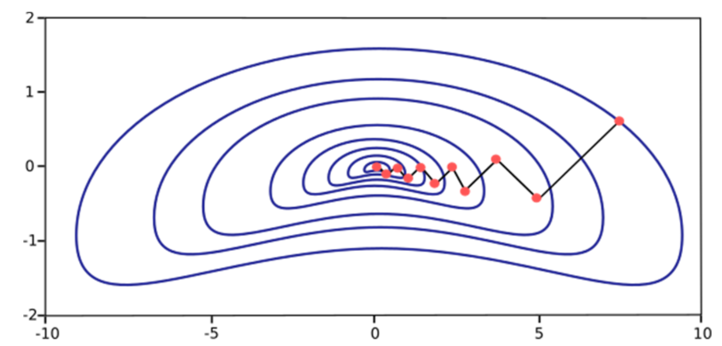
Adapted from https://towardsdatascience.com/

Adaptive Moment Estimation – Adam

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$

…. and beyond



Adaptive Moment Estimation (Adam) – adaptive moment & learning rate

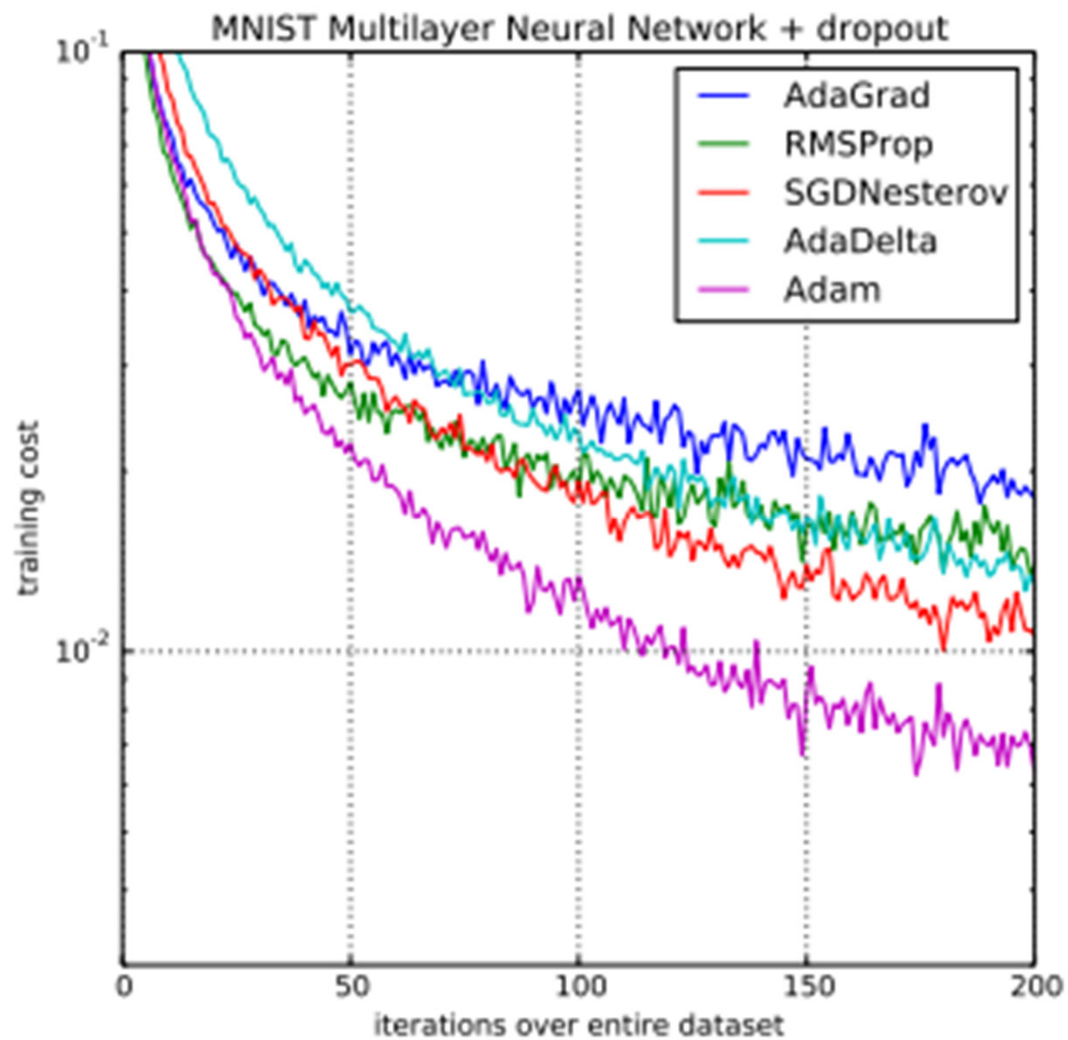
combination of *AdaDelta* (exponentially decaying average of past squared gradients) and *momentum* (exponentially decaying average of past gradients)

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \varepsilon \frac{\mathbf{v}^{(i)}}{\delta + \sqrt{\mathbf{r}^{(i)}}}$$

$$\mathbf{v}^{(i+1)} = \rho_1 \mathbf{v}^{(i)} + (1-\rho_1)\mathbf{g}^{(i)}, \quad \mathbf{r}^{(i+1)} = \rho_2 \mathbf{r}^{(i)} + (1-\rho_2)\left(\mathbf{g}^{(i)}\right)^2$$

Adapted from https://towardsdatascience.com/

Comparison of learning algorithms on MNIST

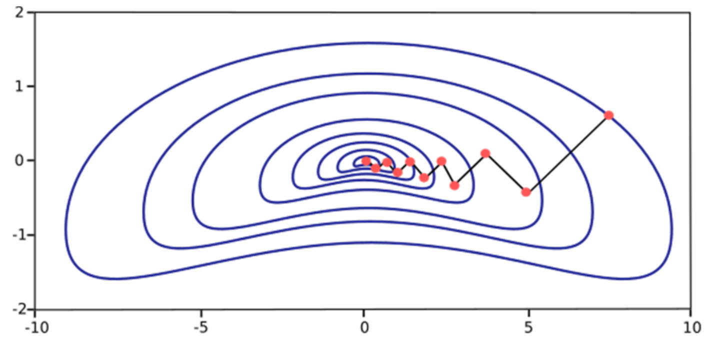


Ruder, 2017: “An overview of gradient descent optimization algorithms”

An old extension of gradient descent – quickprop

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$



…. and beyond

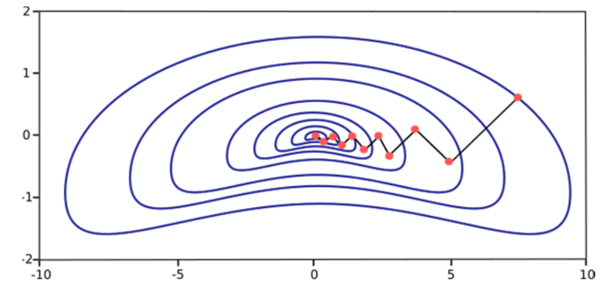
Quickprop (local quadratic approximation)

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \frac{\mathbf{g}^{(i)}}{\mathbf{g}^{(i-1)} - \mathbf{g}^{(i)}}\Delta\mathbf{w}^{(i)}$$

Newton method

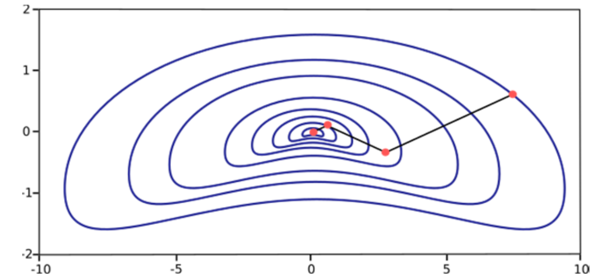
Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$



Newton's method (making explicit use of Hessian, **H**)

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - (\mathbf{H}^{(i)-1} \cdot \mathbf{g}^{(i)})\eta$$

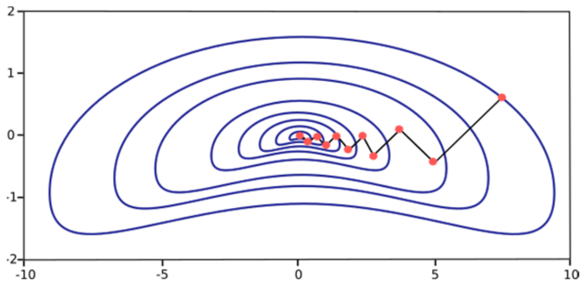


- $O(NW^2)$ to calculate and $O(W^3)$ to inverse Hessian
- away from the optimum, Hessian may not be semi positive definite

Newton method

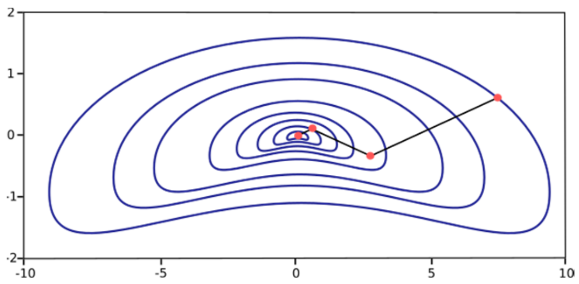
Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$



Newton's method (making explicit use of Hessian, **H**)

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - (\mathbf{H}^{(i)-1} \cdot \mathbf{g}^{(i)})\eta$$



The Levenberg-Marquardt algorithm

- Newton's approach with model trust region (the region where the inverse approximation and error direction with Hessian can be trusted)
- Developed specifically for minimising MSE

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - (\mathbf{J}^{(i)T} \cdot \mathbf{J}^{(i)} + \lambda^{(i)}\mathbf{I})^{-1} \cdot (2\mathbf{J}^{(i)T} \cdot \mathbf{e}^{(i)})$$

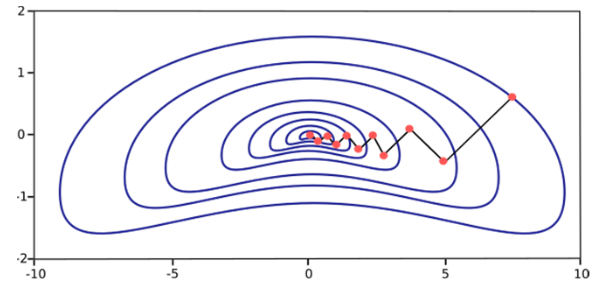
$$\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J}$$

λ: between Newton and standard gradient descent

Newton method

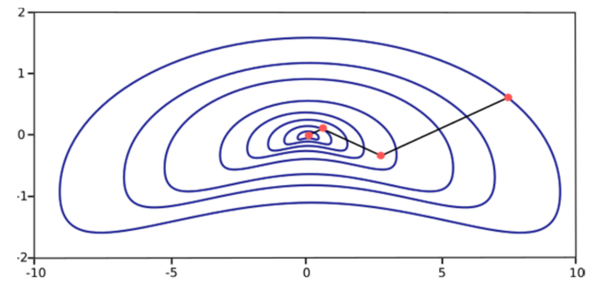
Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$



Newton's method (making explicit use of Hessian, **H**)

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - (\mathbf{H}^{(i)-1} \cdot \mathbf{g}^{(i)})\eta$$



Quasi-Newton approach

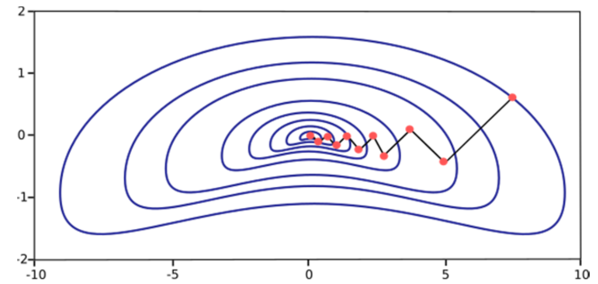
- iterative approximation **G** of the inverse Hessian **H** using first derivative of the error function (e.g. Broyden-Fletcher-Goldfarb-Shanno)

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - (\mathbf{G}^{(i)} \cdot \mathbf{g}^{(i)}) \cdot \eta^{(i)}$$

Conjugate gradient method

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mathbf{g}^{(i)}\eta^{(i)}$$



Conjugate gradient method

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + \mathbf{d}^{(i)} \cdot \eta^{(i)}, \qquad \mathbf{d}^{(i+1)} = \mathbf{g}^{(i+1)} + \mathbf{d}^{(i)} \cdot \gamma^{(i)}$$

- Intermediate approach between gradient descent and Newton method
- training along the conjugate directions regarding the Hessian matrix (without calculating the Hessian)
- γ is a conjugate parameter (has to be estimated)
- It reminds of the minimisation with the momentum term but with adaptive coefficients (incl. learning rate)
- model trust region instead of line search -> *scaled conjugate gradient* method