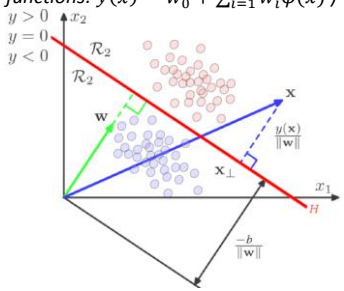
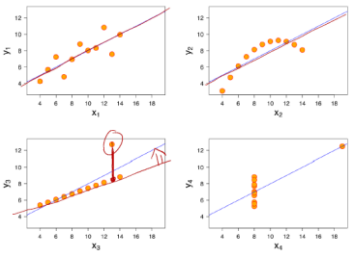
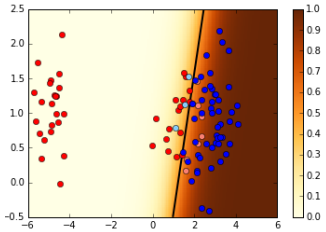
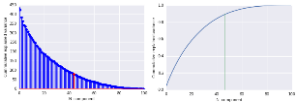
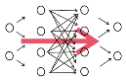
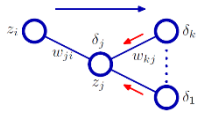
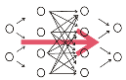
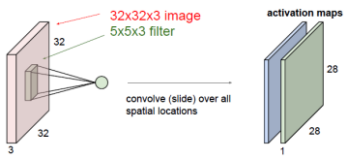
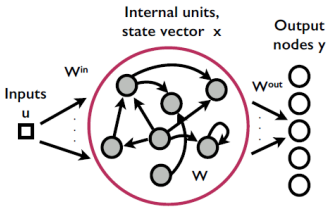
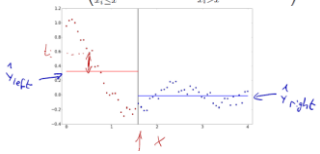
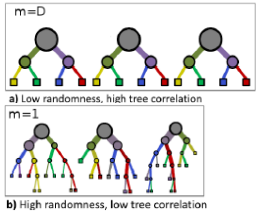
Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Linear Discriminant Analysis (LDA)	ML2.x Bishop 4.x	**Decision function $y(x) = w^T x + b$** *(generalized LDA with non-linear basis functions: $y(x) = w_0 + \sum_{i=1}^{D} w_i \phi(x)$)*  **Fishers Criterion**: maximize distance of projected means, minimize within-class variance s_k^2 of projected classes k $J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \xRightarrow{explicit\ proj.} \frac{w^T S_B w}{w^T S_W w}$ for $argmax_w$: large scatter S_B "between-class" and low scatter S_W "within-class" ***analytic sol.***: $w = S_W^{-1}(m_2 - m_1)$ and $b = -\frac{1}{2} w(m_1 + m_2)$ with $S_W = S_{W_1} + S_{W_2}$	weight vector $\boldsymbol{w} \in R^{Dx1}$ *(perpendicular to decision boundary H, projects data to one dim)* bias/intercept b (or w_0) fixed offset --- ***Free parameters:*** w: $\frac{D\,(D+1)}{2}$ ($w \in R^{DxD}$ is symmetric -> calculate only half + diagonal; with intercept w_0 it is $D + 1$) --- A1: **Gaussian** data distributions for both classes (-> each class can be described by covariance matrix) A2: **equal covariance matrices** of the classes	Supervised Learning Classification Train model with **labelled data** $\langle (x_i, y_i) \rangle_{i=1}^{N}$ to predict **discrete** label y_{N+1} from data x_{N+1} --- • As a first shot method • when noise model of sensors is known • when class means are informative	***training***: $O(D^3)$ for matrix inv. Strassen algorithm $O(D^{2.8})$ for extremely large D / not used in practice: Coppersmith-Winograd method $O(D^{2.3})$ ***test/recall***: $O(D)$	+ weight vector w and bias b can be computed analytically (key: within-class covariance matrices are identical and Gaussian) – covariance matrix needs to be estimated and inverted	Various LDA **formulations** (gradient descent, eigenvalue problem, incremental for online learning) Generalize to **multiple classes**: one-against-rest (bad) one-against-one (better) inherent multiclass formulation: $y_k(x) = w_k^T X + b_k$ **Easier condition:** $S_k = \sigma^2 \mathrm{I}$ (sphere) decision boundary perpendicular to $m_2 - m_1$ search for nearest class mean **Harder condition:** S_k arbitrary normal distr., different per class decision hyperquadratic (instead of hyperplane)
Linear Regression	ML3.x Bishop 3.x	**Regression model $y(x, w) = w^T x + \varepsilon$** $y(x, w) = w_0 + w_1 x_1 + \cdots + w_D x_D + \varepsilon_i$ data point $x = (x_1, \ldots, x_D)^T$, prediction $\hat{y} = w^T x$, residual error ε *(generalized with non-linear basis functions: $y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \phi(x) + \varepsilon_j$)*  **Least Squares Loss Function**: Prediction $\hat{y}(x, w) = w^T X = Xw$ $argmin_w \lVert \varepsilon \rVert^2 = argmin_w \lVert y - \hat{y} \rVert^2$ ***analytic sol.***: $X^T y = X^T X w$ ("normal eq.") $\Leftrightarrow\ w = (X^T X)^{-1} X^T y$ **Regularization** combine Loss functions with penalty for large weight vectors w	weight vector $\boldsymbol{w} \in R^{Dx1}$ *(single weight w_i: determines how sensitive prediction $\hat{y}$ is to change of corresponding input var x_i [partial deriv. w.r.t x_i])* bias/intercept b (or w_0) fixed offset with aug. notation, dim of data is $D + 1$ with $x_0 = 1$ --- ***Free parameters:*** w: $M > \frac{D\,(D+1)}{2}$ --- A1: residuals means: $\forall i: E(\varepsilon_i) = 0$ *↯ offset/skew/outliers/compare A1* A2: Residuals uncorrelated and share same var: $\forall i: Var(\varepsilon_i) = \sigma^2$ *↯ heteroscedastic data (e.g. % err.)* A3: Residuals normally distributed $\varepsilon_i \sim N(0, \sigma^2)$ (with zero mean -> A1) *↯ offset/skew/outliers/compare A1*	Supervised Learning Regression Train model with **labelled data** $\langle (x_i, y_i) \rangle_{i=1}^{N}$ to predict **continuous** label y_{N+1} from data x_{N+1} --- • predict unknown values y_i for new x_i • estimate influence of single/several input var. (est. strength of corr. betw. x_i and y • Visualize relationships	***training***: matrix inv $O(D^3)$ (or $O(D^{2.3})$) ***test/recall***: $O(D)$	+ useful analytical / computational properties – practical applicability limited by curse of dim / linearity	

Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Logistic Regression	ML4.x Bishop4.3.2	wrap output of linear model $z = w^T x$ with sigmoid/logistic function $g(z) = \frac{1}{1+e^{-z}}$: **Model** $\boldsymbol{h_w(x) = g(w^T x) = \frac{1}{1+e^{-w^T x}}}$ output is bound to $0 \leq h_w(x) \leq 1$, can be used as probability whether x belongs to one ($y = 1$) or the other class ($y = 0$ or -1)  **Logistic Cost Function** $\boldsymbol{J(x) = \sum_i^N log(exp(-y_i(w^T x_i)) + 1)}$ use log (quadratic loss $\boldsymbol{J(x)} = \|h_w(x), y\|^2$ is non-convex because of sigmoid) *name: formally derived as regression, but used as classification*	weight vector $\boldsymbol{w} \in R^{Dx1}$ bias/intercept b (or w_0) fixed offset ——————— ***Free parameters:*** w: $M > \frac{D\,(D+1)}{2}$ ——————— A: same as LDA (?)	Supervised Learning Classification (!) Train model with **labelled data** $\langle\langle x_i, y_i\rangle\rangle_{i=1}^N$ to predict **discrete** label y_{N+1} from data x_{N+1} ——————— • if probabilities of label are needed	***training***: $O(D^3)$ ***test/recall***: $O(D)$	+ Convex cost function – There is no analytic solution	**Linear regression** for $g(x) = x$ (identity)
Principal Component Analysis (PCA) singular value decomposition SVD eigenvalue decomposition EVD Karhunen-Loève transform	ML5.x	Project D-dim (R^D) data into a M-dim subspace ($R^M \subset R^D$) with $M \ll D$, that (1) contains the ***relevant part*** of our data and (2) such that data is ***uncorrelated*** (orthogonal basis vectors)  **PCA-Transformation** maximize variance $\sum_{i-1}^N (u_m^T x_i - u_m^T \bar{x})^2 = u_m^T S u_m + \lambda_m(1 - u_m^T u_m)$ (S: data covariance matrix, λ_m: Lagrange multiplier for unit vec constraint $u_m^T u_m = 1$ to avoid $u_m \to \infty$ for $argmax_{u_m} u_m^T S u_m$) ***analytic sol.***: deriv. w.r.t. u_m set to zero: $S u_m = \lambda_m u_m$ (Eigenvalue problem) Take M eigenvectors $u_1, \dots, u_m$ corresp. to the largest eigenvalues $\lambda_1, \dots, \lambda_m$ (sort asc.) e.g. until 99% of variance is preserved	eigenvalue spectrum $\langle\langle \lambda_m, u_m \rangle\rangle_{m=1}^M$ of data covariance matrix $S \in R^{DxD}$ ***hyperparameter***: M: nr. of output dim OR % threshold of cov. explained  ——————— ***Free parameters:*** … ——————— A1: relevance is expressed by variance A2: PCA determines a **linear** subspace	Unsupervised Learning linear feature extractor ——————— pre-process: • data compression • dimensionality reduction	***training***: $O(D^3)$ or iterative methods that scale better for high dim. ***test/recall***: $O(D)$	+ popular – prone to outliers / scaling of a variable (cm instead of m) – doesn't guarantee good class-seperability (unsupervised; doesn't take class labels into acc) – computational complexity of calculating eigenvalues grows cubically in D	For **large dimensionality D**, using covariance matrix has disadvantages ($O(D^3)$). Various PCA **formulations** and (SVD, Bayesian PCA, iterative vs. analytical, Raleigh coeff., …) **Related subspace methods:** Whitening / Sphering: transform data to zero mean and unit covariance (common preprocessing step) Factor analysis (FA): incorporate domain-specic assumptions Canonical correl. analysis (CCA): relate two data sources to a common subspace which maximizes cross-covariance Kernel-PCA non-linear extension of PCA

Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Independent Component Analysis (ICA)	ML5b.x	**Mixture/forward model $\boldsymbol{x} = \boldsymbol{As}$** with both unknown mixing matrix $A \in R^{NxN}$ and sources $s_1, \dots, s_n$ maximize statistical independence (vs. PCA: max. cov.) **backward model $\boldsymbol{s} = \boldsymbol{A^{-1}x} = \boldsymbol{Wx}$** generative (describes how data x is generated by mixing process) **Gradient descent (GD) on w:** $\hat{\boldsymbol{s}} = \boldsymbol{w^T x}$ 1: init w 2: determine direction in which kurtosis of $\hat{\boldsymbol{s}}$ decr./incr. most strongly (for pos/neg kurtosis) 3: run a step with GD for improved w	… _____ ***Free parameters:*** … _____ A1: source signals s_i are independent of each other A2: observed signals x_i are a linear mixing of source signals s_i with a fixed mixing matrix A. A3: sources have a non-Gaussian distribution (one may be Gaussian). Helpful: mean-free data Helpful: whitened data ($A \rightarrow \tilde{A}$) reduce nr of parameters to be estimated from D^2 to $\frac{D(D-1)}{2}$ since	Unsupervised Learning _____ computational method for separating a multivariate signal (e.g. sounds) into (independent) additive subcomponents (e.g. speakers or singers) examples: • audio sources (no reverberation, movement of sources) • EEG sources	***training***: $O(D^3)$	+ many variants (with varying interpretation of "statistical independence" for specific req of data) linear method: + once trained can be applied extremely fast (e.g. online systems) + independent components can be visualized – sources have arbitrary sign, order and amplitude (finding matching components is not trivial)	kurtosis hard to estimate robustly, other measures of non-Gaussianity: (max) negentropy, (min) mutual information, (max) likelihood, … **Related subspace methods:** > compare PCA Blind Source Separation (BSS) ICA is a special case of BSS

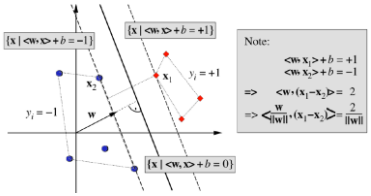
Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Multilayer Perceptron (MLP) **Feed-Forward Neural Network** (NN) 	ML7.x Bishop 5	Make basis functions linear comb. of input with adaptive parameters and train them: Extension of logistic regression model $y = \sigma(w^T \phi(x))$ σ: output activation func. ϕ: basis function with learned params $w^{(l)}$ Network organized in layers, output units of lth layer serve as input of the $l+1^{\text{th}}$ layer **network model:** $\boldsymbol{a_{u_l} = \sum_{u_{l-1}=1}^{D_{l-1}} w_{u_l u_{l-1}}^{(l)} z_{u_{l-1}}}$ $z_{u_l} = h(a_{u_l})$: u_l^{th} unit of layer l $D_l = \dim(z_{u_l})$: nr of hidden units of layer l u_l: idx of units of layer l $w_{u_l u_{l-1}}^{(l)}$: D_l x D_{l-1} weights of layer l $z_{u_0} = x_i$: i^{th} input "unit"/data point $D_0 = \dim(x_i)$: nr of input var. $x_i = (x_1, \dots, x_{D_0})$ $a_{u_2} = y_k$: D_k-Dimensional output in example three-layer network: $y_k = \sigma\left(\sum_{k=0}^{D_2} w_{ku_2}^{(3)} h\left(\sum_{u_1=0}^{D_1} w_{u_2u_1}^{(2)} h\left(\sum_{i=0}^{D_0} w_{u_1 i}^{(1)} x_i\right)\right)\right)$ **generic MLP learning algorithm** 1: choose initial weight vector w 2: intialize minimization approach 3: ***while*** error did not converge ***do*** 4: ***for*** all $(x_n, t_n) \in \mathcal{D}$ do 5: $y_n \leftarrow$ forward pass(x_n) 6: $\frac{\partial E_n}{\partial w_{u_l u_{l-1}}} \leftarrow$ backward pass(y_n, t_n) 7: ***end*** *for* 8: sum over all n errors $\sum_n \frac{\partial E_n}{\partial w_{u_l u_{l-1}}}$ 9: perform update step 10: %one epoch: all data considered once% 11: ***end*** *while* **training:** with data $\mathcal{D} = \langle\langle x_n, t_n\rangle\rangle_{n=1}^N$ $\text{argmin}_w[\text{E}(\text{w}, \mathcal{D}) = \sum_{n=1}^N E_n(w)]$ **backpropagation** (chain rule) partial derivative of error E_n w.r.t weight $w_{u_l u_{l-1}}$: $\frac{\partial E_n}{\partial w_{u_l u_{l-1}}} = \frac{\partial E_n}{\partial a_{u_l}} \frac{\partial a_{u_l}}{\partial w_{u_l u_{l-1}}} = \delta_{u_l} z_{u_{l-1}}$ for output unit y_k: $\delta_k = \frac{\partial E_n}{\partial a_k} = \frac{\partial E_n}{\partial y_k}\frac{\partial y_k}{\partial a_k} = y_k - t_k$ for hidden unit z_{u_l}: $\delta_{u_l} = \frac{\partial E_n}{\partial a_{u_l}} = \sum_{u_{l+1}} \frac{\partial E_n}{\partial a_{u_{l+1}}} \frac{\partial a_{u_{l+1}}}{\partial a_{u_l}} = \sum_{u_{l+1}} \frac{\partial E_n}{\partial a_{u_{l+1}}} \frac{\partial a_{u_{l+1}}}{\partial z_{u_l}} \frac{\partial z_{u_l}}{\partial a_{u_l}} = \sum_{u_{l+1}} \delta_{u_{l+1}} w_{u_{l+1}u_l}^{(l+1)} h'(a_{u_l})$ 	***nonlin. Activation Functions*** $\boldsymbol{h(a_j)}$: transform activations of lin. comb. $\boldsymbol{h_{linear}(a_j) = a_j}$ $h'(a_j) = 1$ $\boldsymbol{h_{logistic}(a_j) = \frac{1}{1+exp(-a_j)}}$ $h'(a_j) = h(a_j)(1 - h(a_j))$ $\boldsymbol{h_{tanh}(a_j) = \frac{e^{a_j} - e^{-a_j}}{e^{a_j} + e^{-a_j}}}$ $h'(a_j) = 1 - h(a_j)^2$ $\boldsymbol{h_{relu}(a_j) = \max(0, a_j)}$ $h'_{a_j<0}(a_j) = 0 \vee h'_{a_j \geq 0}(a_j) = 1$ ***pairs of output act. and error func.***: binary classification: $\sigma_{logistic}(a)$ prob. interpret. $0 \leq y(x,w) \leq 1$: $p(C_1\|x)$ Cross-entropy error function (neg. log likelih.) $E(w) = -\sum_n (t_n \ln y_n + (1-t_n)\ln(1-y_n))$ regression: $\sigma_{linear}(a)$ $E(w) = -\sum_{n=1}^N (y_n - t_n)^2$ multiclass classification: generalization of cross-entropy err.: $E(w) = -\sum_{n=1}^N \sum_{k=1}^K (t_{kn} \ln y_k(x_n, w))$ **forward pass** 1: ***for*** all input units i ***do*** 2: set $z_{u_0} = x_i(n)$ 3: ***end*** *for* 4: ***for*** all hidden layers l (forward) ***do*** 5: ***for*** all units u_l in layer l ***do*** 6: set $a_{u_l} = \sum_{u_{l-1}=1}^{D_{l-1}} w_{u_l u_{l-1}}^{(l)} z_{u_{l-1}}$ 7: set $z_{u_l} = h(a_{u_L})$ 8: ***end*** *for* 9: ***end*** *for* 10: ***for*** all output units k ***do*** 11: set $a_{u_L} = \sum_{k=0}^{D_k} w_{ku_L}^{(L)} z_{u_L}$ 12: set $y_k = \sigma(a_{u_L})$ 13: ***end*** *for* ***Free parameters:*** $\sum_{l=1}^L D_l$ x D_{l-1} $D_0 = D_{in} = \dim(x_i)$ and $D_L = D_{out} = \dim(y_k)$ for two layers: D_{in} x $D_1 + D_1$ x D_{out} A1: error function is continuous A2: error function is differentiable	Supervised Learning Nonlinear Regression / Classification / Multiclass Classification (K output units and softmax activation to $y_k = \frac{\exp(a_k(x,w))}{\sum_j \exp(a_j(x,w))}$) • Representation Learning: Parametrize basis functions and adapt the data to discover useful representations automatically **backward pass** 1: ***for*** all output units k ***do*** 2: compute $\delta_k = y_k - t_k$ 3: ***end*** *for* 4: ***for*** all hidden layers l (backward) ***do*** 5: ***for*** all units u_l in layer l ***do*** 6: backpropagate existing δ's to obtain δ_{u_l} 7: compute $\frac{\partial E_n}{\partial w_{u_l u_{l-1}}} = \delta_{u_l} z_{u_{l-1}}$ 8: ***end*** *for* 9: ***end*** *for*	***training***: [feedforward & backward pass] * nr. of gradient descent steps ***test/recall***: feedforward pass	+ many applications resulting model can be significantly more compact (than SVM with same generalization prop) -> faster to evaluate – price for compactness: no convex likelihood (optimization) function – not robust to hyperparameter settings	**Perceptron** uses step-function as non-linearities (hence nondifferentiable) **two layer example** first (hidden) layer: $a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i$ and $z_j = h(a_j)$ second (output) layer: $a_k = \sum_{j=0}^M w_{kj}^{(2)} z_j$ and $y_k = h(a_k)$ => $y_k = \sigma\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right)$ **three layer example** first (hidden) layer: $a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i$ and $z_j = h(a_j)$ second (hidden) layer: $a_p = \sum_{j=0}^M w_{pj}^{(2)} z_j$ and $z_p = h(a_p)$ third (output) layer: $a_k = \sum_{p=0}^P w_{kp}^{(3)} z_p$ and $y_k = h(a_k)$ **Matrix notation:** $a_{u_l} = W^{(l)} z_{u_{l-1}}$ a_{u_l}: D_l-sized vector of linearly combined units from previous vector: $a_{1_l} = w_{1_l 1_{l-1}} z_{1_{l-1}} + w_{1_l 2_{l-1}} z_{2_{l-1}} + \cdots + w_{1_l D_{l-1}} z_{D_{l-1}}$ $a_{D_l} = w_{D_l 1_{l-1}} z_{1_{l-1}} + w_{D_l 2_{l-1}} z_{2_{l-1}} + \cdots + w_{D_l D_{l-1}} z_{D_{l-1}}$ $W^{(l)}$: a D_l x D_{l-1} weight matrix $z_{u_{l-1}}$: D_{l-1}-sized vector of hidden units **Alternative indices:** Each layer has hidden units $z_{l_{out}} = h(a_{l_{out}})$ and a lin. comb. of input $z_{l_{in}}$ with weights $w_{l_{out} l_{in}}^{(layer)}$: $a_{l_{out}} = \sum_{l_{in}=1}^{L_{in}} w_{l_{out} l_{in}}^{(layer)} z_{l_{in}}$

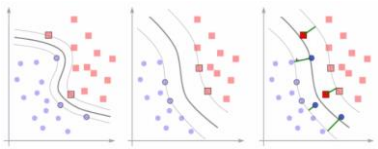
Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Conv. Neural Networks (CNN) 	ML11.x	**Convolution**: filter w: w^T + b  slide filter over image and compute (5x5x3 + bias) dot product at each location (5x5x3) filter weights are shared across activations of one activation map often implemented as cross-correlation (difference in mathematics) **Nonlinearity**: nowadays usually ReLU **Pooling**: (~subsampling) (not trained) makes representations smaller and more manageable; operates over each activation map independently; pooling filter size makes representation approximately invariant to small translations in the input nowadays usually max-pooling important to keep convolution cost low **Regularization** through weight sharing	working with volumes (instead of vectors) ***hyperparameters:*** **stride**: how fine grained you want to sample input layer **Layer output size**: $\left(\frac{(N-F)}{stride}\right) + 1$ Layer input dim N, Filter dim F ***valid convolution***: no padding and kernel required to be fully contained in image (but decreases output volume) ***same convolution***: with padding to keep image size constant through successive convolutions ***zero padding:*** with $\frac{F-1}{2}$ (to preserve image size) ***Free parameters:*** $\sum_{l=1}^{L} \ldots$ ***nr of parameters per layer:*** filter dimensions * output dimension: $[F \times F \times N_{layer\ l-1} + 1\ (= bias)] * N\_layer1$	unsupervised Learning Nonlinear Regression / Classification / Multiclass Classification (K output units and softmax activation to $y_k = \frac{\exp(a_k(x,w))}{\sum_j \exp(a_j(x,w))}$) • Image Classification / Semantic Segmentation / Retrieval / Captioning / … • Pose estimation / view generation / … • Playing games	***training***: [feedforward & backward pass] * nr. of gradient descent steps ***test/recall***: feedforward pass	+ tremendously successful in practical applications + shared parameters (built-in regularization) – expensive training – many hyperparameters / complex tuning – not robust to hyperparameter settings	Recurrent Neural Network (RNN) Multilayer Perceptron (MLP) fully connected version
Recurrent Neural Network (RNN) 		**Allow for cycles in connectivity graph** input u(t) x(t) = f(net^x(t)) [nonlinearity f] net^x(t) = Wx(t-1) + W^in u(t) [recurrent weights W, input weights W^in]  output y(t) = f^out(net^y(t)) [f^out output activation function] net^y(t) = W^out x(t) **Backpropagation Through Time (BPTT)** Idea: unfold network over time (results in one very deep network), then do backpropagation with weight sharing (at each step the same weights are used)	***BPTT problems:*** error gradient is propagated back and multiplied with weight w and derivative of unit's activity f'(net) **exploding gradient:** If |f'(net)w > 1| for all t, exponential increase => oscillating weights **vanishing gradient:** If |f'(net)w < 1| for all t, exponential decrease => slow convergence => cure: LSTM	unsupervised Learning Nonlinear Regression / (Multiclass) Classification (K output units and softmax activation to $y_k = \frac{\exp(a_k(x,w))}{\sum_j \exp(a_j(x,w))}$) • processing sequences • dynamical systems (rather than function mappings)		+ tremendously successful in practical applications + shared parameters (built-in regularization) + arbitrary complex – expensive to train – many hyperparameters / complex tuning – not robust to hyperparameter settings – vanishing gradient → LSTM – exploding gradient	

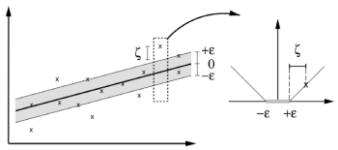
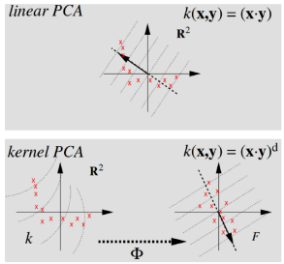
Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Classification and Regression trees (CART)	ML8.x (regression) ML8.21ff (classification)	CART(X, t, `max_depth`, `min_leaf`) 1: Check whether data should be split further; otherwise return leaf node 2: Find best split value for each feature → Greedily minimize sum of squared errors in the two children $x_{\text{split}} = \arg\min_x \left(\sum_{x_i \le x} (t_i - \hat{y}_{left})^2 + \sum_{x_i > x} (t_i - \hat{y}_{right})^2 \right)$  3: Choose best combination of split feature and split value (w.r.t. to squared error) 4: Split data into left and right accordingly: (X_l, t_l) and (X_r, t_r) 5: Save split feature and value, and pointer to two new subtrees to be 6: built recursively: (CART(X_l ,t_l, `max_depth`-1, `min_leaf`), CART(X_r, t_r, `max_depth`-1, `min_leaf`))	Input: data X, targets t ***hyperparameters*** `min_leaf`: nr of samples (in leaf) `max_depth`: of the tree Total number of nodes Split model weak learner: axis-aligned splits -> constant, other split models) Leaf/split model (majority vote; probability of data in the leaf) Split criterion Gini index; variance reduction; information gain **entropy** "disorder/uncertainty" of Random Var. V with K possible outcomes v_k and distr $p(v_k)$ $H(V) = -\sum_{k=1}^{K} p(v_k) \log_2 p(v_k)$ **information gain**: V has N data points with; split s into nodes N_l and N_r, RVs V_l and V_r with p_l and p_r: $I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$ High-variance, low bias model (small changes of data may result in very different trees) _____ ***Free parameters:*** ... _____	supervised Learning unsupervised Learning Nonlinear Regression / (Multiclass) Classification _____ • semi-supervised learning • density estimation • embedding learning	***training***: ... ***test/recall***: ... ***storage during / after building***: $O(N)$ / $O(ND)$ N data points of dimensionality D, axis-aligned splits	+ easy to interpret + directly handle categorical features + scalable to large datasets (fast) + flexible framework with exchangeable components (split criterion, leaf model, type of split) – Tend to overfit – deterministic (i.e. not suitable for some ensemble methods) High-variance, low bias model (small changes of data may result in very different trees) random forests are said to be the best off-the-shelf model (for many applications)	
Random Forests		RF for CART 1: For b = 1 to B 2: Draw bootstrap sample Z* of size N from training data 3: Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_min is reached: 4: select m variables at random from p variables 5: pick the best variable/split-point among the m 6: split the node into two daughter nodes 7: Output the ensemble of trees {T_b}_1^B prediction for new point: Regression: f^_rf^B(x) = 1/B sum[b=1:B] T_b(x) Classification: Let C^_b(x) be the class pred. of the b-th random-forest tree. Then C^_rf(x) = majority vote {C^_b(x)}_1^B	***hyperparameters*** CART-hyperparameters, additional: m: nr of (randomly choosen) variables at each split; B: Total number of trees _____ ***Free parameters:*** ... _____ A1: individual models perform (reasonably) well A2: different models are not correlated	supervised Learning unsupervised Learning Nonlinear Regression / (Multiclass) Classification _____ Motivation: Ensemble error depends on: • strength of indiv. models (-> we want indiv. good working models) • uncorrelatedness of models' errors (-> ... with high variance)	***training***: ... ***test/recall***: ... Best case: balanced trees Fitting: O(BmN logN) Prediction: O(B logN) Worst case: splitting o one data point at a time Fitting: O(BmN2) Prediction: O(BN) ***storage during / after building***: $O(N)$ / $O(ND)$ N data points of dimensionality D, axis-aligned splits	+ robust to hyperparameter settings (vs. neural networks) + robust performance even for small datasets + bootstrapping: robust to outliers (and decorrelated trees) – Relatively weak performance for smooth functions without noise Out-of-bag error with little overhead: not every data point is used to t every single tree in fact, almost 37% are not used in each tree (see assignment) predict unused points for each tree to get unbiased estimated of the generalization error	 a) Low randomness, high tree correlation b) High randomness, low tree correlation Decision Tree + (best of a fixed nr of) random splits = **ExtraTree** Extra Tree + Bagging = **ExtraTrees** Decision Trees + Bagging = **Bagged trees** Decision Trees + best split using a random subset of m <= D features + Bagging = **Random forest**

Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Boosting / AdaBoost	ML6.36 ML9.x	Ensemble technique, submodels are trained sequentially, with the m-th submodel trained to fix the mistakes of the previous (first m-1) submodels AdaBoost Binary classication class, labels +1 and -1 Final model $G(x)$ combines individual submodels $G_1, \dots, G_M$, through a weighted majority vote with weights $\alpha_1, \dots, \alpha_M$: $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$ with weights $\alpha_m(err_m) = \log(\frac{1-err_m}{err_m})$ $(\alpha_m(0.5) = 0,\ 0 \leq err_m \leq 1 \rightarrow \alpha_m \rightarrow \pm 10)$ weighted training error rate: $err_m = \frac{\sum_{i=1}^{N} w_i^m I(y_i \neq G(x_i))}{\sum_{i=1}^{N} w_i^m}$ with $w_i^{m+1} = w_i^m \cdot \exp\left(\alpha_m I(y_i \neq G(x_i))\right)$ being adapted weights for each iteration m and indicator function I: $I(a) = \begin{cases} 1, & if\ a = true \quad \Rightarrow \quad w_i^m \\ 0, & else \quad \Rightarrow \quad w_i^m \cdot \exp(\alpha_m) \end{cases} = w_i^{m+1}$ AdaBoost algorithm (weights w_i^m don't need to be stored, may be overwritten) 1: init weights $w_i^0 = \frac{1}{N}, i = 1, 2, \dots, N$ 2: for $m = 1\ to\ M$: 3: fit classifier $G_m(x)$ to w_i^{m-1}-weighted train data 4: compute $err_m = \frac{\sum_{i=1}^{N} w_i^m I(y_i \neq G(x_i))}{\sum_{i=1}^{N} w_i^m}$ 5: compute $\alpha_m(err_m) = \log(\frac{1-err_m}{err_m})$ 6: set $w_i^{m+1} = w_i^m \cdot \exp\left(\alpha_m I(y_i \neq G(x_i))\right)$ 7: output $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$ 	***hyperparameters*** CART-hyperparameters, additional: m: nr of (randomly choosen) variables at each split; B: Total number of trees ***Free parameters:*** ...			+ good generalization capabilities on low noise data – if data is very noisy, it may overfit badly successfully used in practice boosting (like RF) said to be the best off-the-shelf model (for many applications)	Ensemble technique like bagging, but models are not trained independently
Forward Stagewise Additive Modelling (FSAM)	ML9.25ff	loss function L(. , .): quantify error between our model predictions f(x_i) and targets yi E.g., squared error: L(yi; f(xi))=1/2(yi - f(xi))2 way to construct a model to greedily minimize a loss function - E.g., a routine to t the parameters of some model b: m = argmin_> sum[i=1:N] L(yi; b(xi;))				+	can be shown to recover the AdaBoost algorithm.

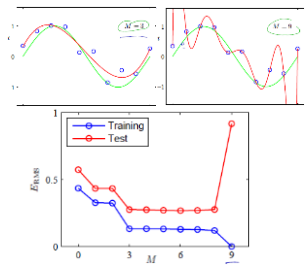
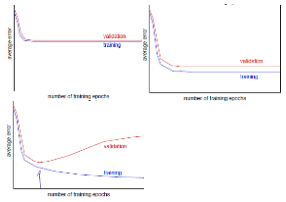
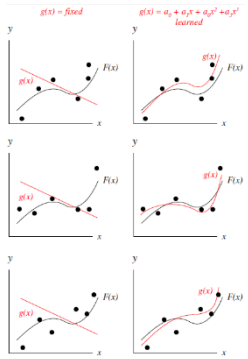
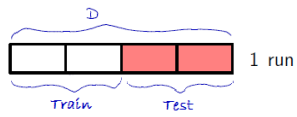
Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Gradient Tree Boosting	ML9.30ff	minimize this: $L(f) = \sum_{i=1}^{N} L\big(y_i, f(x_i)\big)$ w.r.t. f (constraint A2) One can see minimization of training error as gradient descent in the N-dimensional space of "parameters" that describe the values of f at the N data points x_i $\hat{f} = argmin_f\big(L(f)\big)$ with $f = [f(x_1), \dots, f(x_N)]^T$ Gradient Tree Boosting Algorithm 1: init weights $f_0(x) = argmin_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$ 2: for $m = 1\ to\ M$: 3: for $i = 1, 2, \dots, N$ compute $r_{im} = -\left[\left(\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right)\right]_{f=f_{m-1}}$ 4: fit a regression tree to targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \dots, J_m$ 5: for $i = 1, 2, \dots, N$ compute $\gamma_{jm} = argmin_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$ 6: update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ 7: output $\hat{f}(x) = f_M(x)$	_____ ***Free parameters:*** … _____ A1: Loss function is differentiable A2: f comes from a certain model class (e.g., a sum of trees)	supervised Learning unsupervised Learning Nonlinear Regression / (Multiclass) Classification _____ Motivation: • computationally effective boosting methods only for exp/squared error loss • With gradient boosting: any differentiable loss function works	***training***: $O_{best}(BmN\ logN)$ $O_{worst}(BmN^2)$ ***test/recall***: $O_{best}(B\ logN)$ $O_{worst}(BN)$ ***Best case:*** balanced trees ***Worst case:*** splitting one data point at a time ***storage during / after building***: $O(N)$ / $O(ND)$ N data points of dimensionality D, axis-aligned splits	+ robust to hyperparameter settings (vs. neural networks) + robust performance even for small datasets + bootstrapping: robust to outliers (and decorrelated trees) – Relatively weak performance for smooth functions without noise Out-of-bag error with little overhead: not every data point is used to t every single tree in fact, almost 37% are not used in each tree (see assignment) predict unused points for each tree to get unbiased estimated of the generalization error	

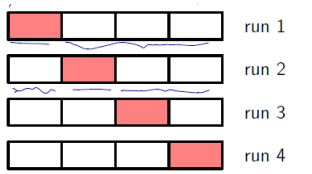
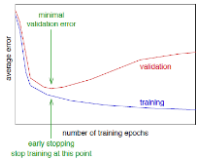
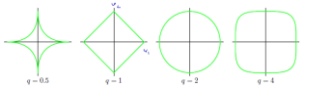
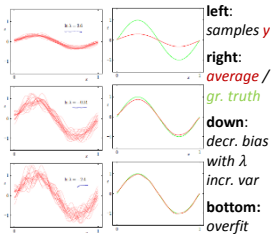
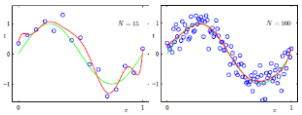
Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Large Margin classifier Optimal hyperplane estimator	ML13.x	 **Large Margin Classifier** optimize w, b to get the largest possible smallest distance between data points and decision hyperplane x: $argmax_{w\in\mathcal{H},b\in\mathbb{R}} \min(\|x - x_i\|)$ for $i = 1, \dots, m$ with $x \in \mathcal{H}$ and $\langle w, x\rangle + b = 0$ (decision function) This is solved by the shortest normal vector w (that still gives correct classification results) $argmin_{w\in\mathcal{H},b\in\mathbb{R}} \tau(w) = \frac{1}{2}\|w\|^2$ (**objective function** τ) subject to **inequality constraint** $y_i(\langle w, x_i\rangle + b) \geq 1$ with $i = 1, \dots, m$ (≥ 1fixes the scaling of w, data should be a bit away from decision plane, 1 could be any other number) **Lagrangian**: first part should be small, second part large $$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i y_i(\langle w, x_i\rangle + b) - 1$$ approach: use partial derivatives, minimize Lagrangian L w.r.t. primal variables w and b and maximize w.r.t. dual variables α_i (effectively finds solution in a saddle point) $\frac{\partial}{\partial b} L(w, b, \alpha) = 0 \rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0$ $\frac{\partial}{\partial w} L(w, b, \alpha) = 0 \rightarrow \sum_{i=1}^{m} \alpha_i y_i x_i = w$ eliminate primal variables by substitution (of w and b) dual optimization problem of vector α (quadratic program): $$argmax_{\alpha\in\mathbb{R}^m} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle$$ subject to $\alpha_i \geq 0$ for all $i = 1, \dots, m$ and $\sum_{i=1}^{m} \alpha_i y_i = 0$ hyperplane decision function “instance based learning” $$f(x) = sign\left(\sum_{i=1}^{m} \alpha_i y_i \langle x, x_i\rangle + b\right)$$ just working on dot products of data. Vector α may be sparse: a lot of $\alpha_i = 0$, corresponding x_i do not contribute to decision hyperplane. Otherwise they are called Support Vectors (SV).		supervised Learning Classification ***linearly seperable case*** _____ Motivation: • basis for SVM		+ just working on dot products, no assumptions about data vs + maximizing the margin also restricts the function class (simple/smooth functions are preferred) – …	instance based learning: **k-Nearest Neighbor** (drawback: save all data points, here: only ones with “active” lagrange multipliers α_i that actually influence the decision hyperplane; corresponding training data points are called support vectors)

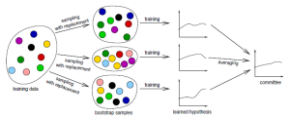
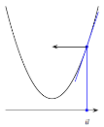
Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Support Vector Machines (SVM)	ML13.x ML14.x Bishop 7	See **Large Margin classifier** (generalization by kernel trick): optimize w, b to get the largest possible smallest distance between data points and decision hyperplane x in non-linear space $\boldsymbol{argmax}_{\alpha \in \mathbb{R}^m} \boldsymbol{W(\alpha)} = \sum_{i=1}^m \boldsymbol{\alpha_i} - \frac{1}{2}\sum_{i,j=1}^m \boldsymbol{\alpha_i \alpha_j y_i y_j k(x_i, x_j)}$ subject to $\alpha_i \geq 0$ for all $i = 1, \dots, m$ and $\sum_{i=1}^m \alpha_i y_i = 0$ **Building Blocks:** **similarity measure k / "kernel"** for: arbitrary input space, e.g. set $x \in \mathcal{X}$ (previously we had: $x \in \mathbb{R}^N$ vector space with defined similarity) (1) delivers a real number describing similarity k between two patterns $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}^N, (x, x') \to k(x, x')$ (2) is symmetric $k(x, x') = k(x', x)$) define from dot product of a feature space $\mathcal{H} \in \mathbb{R}^N$: $k(x', x) := \langle x, x' \rangle = \langle \phi(x), \phi(x') \rangle$ where $\phi: \mathcal{X} \to \mathcal{H}$ maps **input space $\mathcal{X}$** to **feature space $\mathcal{H}$** **Large margin classifier** optimize w, b to get the largest possible smallest distance between data points and decision hyperplane x linear case: $f(x) = sign\left(\sum_{i=1}^m \alpha_i y_i \langle x_i, x_j \rangle + b\right)$ with kernel trick: $f(x) = sign\left(\sum_{i=1}^m \alpha_i y_i \langle \phi(x_i), \phi(x_j) \rangle + b\right)$ $= sign\left(\sum_{i=1}^m \alpha_i y_i k(x, x_i) + b\right)$	**hyperparameters:** kernel parameters: polynomial kernel (param d) $k(x', x) = \langle x, x' \rangle^d$ Gaussian radial basis functions (param σ) $k(x', x) = \exp\left(-\frac{1}{\sigma^2}\|x, x'\|\right)$ sigmoid kernel (param $k > 0$ and $\theta < 0$) $k(x', x) = \tanh(k\langle x, x' \rangle + \theta)$ _____ ***Free parameters:*** ... _____ A1: All computations can be formulated in dot product space **VC bound** VC dimension h of function class: number of data points m you can scatter in feature space $\mathbb{R}^N$ (proportional to capacity) $R[f] \leq R_{emp}[f] + \sqrt{\frac{1}{m}\left(h\left(\log\frac{2m}{h} + 1\right) - \log\frac{\delta}{4}\right)}$ risk is bounded by training error + capacity risk w.r.t. to observed data ("empirical risk" / "average training error"): $R_{emp}[f] = \frac{1}{m}\sum_{i=1}^M (\frac{1}{2}\lvert f(x_i) - y_i \rvert)$ risk w.r.t. to underlying data distribution: $R[f] = \int \frac{1}{2}\lvert f(x) - y \rvert \; dP(x, y)$	supervised Learning Classification _____ Motivation: • make use of nonlinear mappings but don't compute dot product in feature space explicitly	***training***: ... ***test/recall***: ...	+ good in large feature space + very robust performance, even when used as black box + very few hyperparameters + inspecting selected support vectors may help to understand the problem + SVM formulation can be extended to non-seperable cases (in feature space) → soft-margin SVM + SVM regression and outlier detection ("one-class SVM") easily obtained + dealing with unbalanced classes by weighting influence of missclassifications separately per class – kernel computations expensive for many training data points → chunking methods avoid calculating full kernel matrix – bad inspection properties (vs. linear methods) – outperformed by Deep NN (at least if dataset is huge)	Relevance SVM (Bishop 7.2) soft-margin SVM slack variables $\xi_i \geq 0$ per data point $y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i$ $argmin_{w \in \mathcal{H}, b \in \mathbb{R}} \tau(w) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^m \xi_i$ hyperparameter C: trade-off between enlarging the margin and minimizing training error
Soft-margin SVM	ML14.x	$\boldsymbol{argmax}_{\alpha \in \mathbb{R}^m} \boldsymbol{W(\alpha)} = \sum_{i=1}^m \boldsymbol{\alpha_i} - \frac{1}{2}\sum_{i,j=1}^m \boldsymbol{\alpha_i \alpha_j y_i y_j k(x_i, x_j)}$ subject to $\boldsymbol{0 \leq \alpha_i \leq C}$ for all $i = 1, \dots, m$ and $\sum_{i=1}^m \alpha_i y_i = 0$ slack variables $\xi_i \geq 0$ per data point $y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i$ $argmin_{w \in \mathcal{H}, b \in \mathbb{R}} \tau(w) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^m \xi_i$ allows for slight violation (penalty must be determined at training time with regularization parameter C resp. ν)  1 – hard margin (complex hyperplane but no errors) 2 – without original SVM 3 – soft margin SVM (potentially smoother hyperplane)	same as SVM $\boldsymbol{b}$ (bias scalar): can be computed exploiting that for all SVs x_i with $\alpha_i < C \to \xi_i = 0$. Intuition: shifts the decision hyperplane such, that SVs with zero slack lie on ± 1 lines. **hyperparameters:** $\boldsymbol{C}$: trade-off between enlarging the margin and minimizing training error ***small C*** *-> may need more SV to define hyperplane (since they have small influence)* ***large C*** *-> fewer SV but with large weights* $\boldsymbol{\nu}$: alternative parametrization with $0 < \boldsymbol{\nu} \leq 1$: bounds the fraction of training patterns which will become SVs and those which will have non-zero slacks. _____ ***Free parameters:*** ... _____	supervised Learning Classification _____ Motivation: • noisy data • misclassified data	***training***: ... ***test/recall***: ...	same as SVM Regularization: + allows solutions with function class of lower capacity (better generalization) + enlarged margin + reduce $R[f]$ – outperformed by Deep NN (at least if dataset is huge)	

Method	slides	Idea	Characteristics / Parameters / Assumptions	Usage / Motivation	Runtime	Pro + and Contra –	special cases / improvements / related methods
Support Vector Regression (SVR)	ML14.27ff	a support vector regression model can estimate continuous non-linear functions $\mathbb{R}^N \to \mathbb{R}$ Idea: ε-tube around function Key ingredients: Kernel trick Use the ε-insensitive loss function (corresponds to regularizing the solution similar to a large margin). Introduce (two types) of slack variables ξ_i, which allow for violations of theε-tube. Limit influence of outliers by introducing a constant C. Choose constans $C, \varepsilon \geq 0$ a priori Formulation via Lagrange multipliers, solve quadratic problem. 		supervised Learning (non-linear) Regression _____			http://www.svms.org/regression/SmSc98.pdf
Kernel Principal Component Analysis (kPCA)	ML14.32ff	Idea: Formulate all calculations of linear PCA (eigenvalue problem!) via dot products of the input space X. obtain a non-linear version of linear PCA by mappings - from input space X to a high-dimensional feature space H! use the kernel trick, thus compute everything in input space X! $$f_n(x) = \sum_{i=1}^{m} \alpha_i^n \, k(x_i, x)$$ where α_i^n are (up to a normalizing constant) the components of the nth eigenvector of the Gram Matrix $K_{ij} := (k(x_i, x_j))$ with $dim(K)$: $M \times M$ and $n = 1, \dots, m$ (as many as data points)  **input space $\mathcal{X}$ to feature space $\mathcal{H}$**	***hyperparameter***: M: nr. of output dim _____ ***Free parameters:*** … _____ A1: …	non-linear feature extractor _____ pre-process for non-kernel methods: • data compression • dimensionality reduction introspection: • into SVM solution: posthoc vis. of variance isolines (of feature func. in $\mathcal{X}$ using SVM hyperparams) • why a classification problem is hard ([1] intrinsically complex/high dim or [2] simple but noisy?)	***training***: $O(N^3)$ (N, M or m: nr of data points) or iterative methods that scale better for high dim. ***test/recall***: …	+ allows non-linear feature reduction – computation of Gram matrix expensive for large nr of patterns	

2. General Machine Learning Concepts

Concept	slides	Idea	Characteristics / Parameters / Assumptions / Examples	Usage / Motivation		special cases / related methods / improvements
Approximat.-Generaliz. Tradeoff	ML6.8	**Approximation-Generalization Tradeoff** more complex <> more flexibility to approx.. less complex <> more likely to generalize squared distance error: $E(w) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, w) - t)^2$ root-mean-square (RMSE) normalized error $E_{RMS}(w) = \sqrt{2E(w^*)/N}$ N: allows to compare different sizes of data sqrt: makes sure E_RMS is measured in same scale as t				
Loss function	ML6.11	expected loss (regression with squared error) $\mathbb{E}[L] = \int (y(x) - h(x))^2 p(x)dx + \int (h(x) - t)^2 p(x,t)dx$ first term: how much does the prediction function ("hypothesis") $y(x)$ differ from the ideal (and theoretical) prediction function $h(x)$ second term: how much does the ideal prediction function differ from the real target labels t. This is the noise inherent to the data.		Supervised Learning _____		
Bias-Variance	ML6.12ff	$\mathbb{E}_{\mathcal{D}}\left[(y(x,\mathcal{D}) - h(x))^2\right] = (\bar{y}(x) - h(x))^2 + \mathbb{E}_{\mathcal{D}}\left[(y(x,\mathcal{D}) - \bar{y}(x))^2\right]$ average performance of *prediction function ("hypothesis")* $y(x,\mathcal{D})$ over multiple *datasets* $\mathcal{D}$; extended and reformulated with *average prediction function* $\bar{y}(x) = \mathbb{E}_{\mathcal{D}}[y(x,\mathcal{D})]$. **(bias)²**: how much differs the *(average) hypothesis (over all datasets)* $\bar{y}(x)$ from the *ideal prediction function* $h(x)$ **variance**: difference of a single hypothesis to the average hypothesis; how much does the model vary with particular data $\mathcal{D}$ **Bias-Variance Tradeoff** ~ Approximation-Generalization Tradeoff less complex <> more likely to generalize (**high bias**, low var) more complex <> more flexibility to approx.. (low bias, **high var**) Balancing bias and variance gives optimal predictive capability.	 left: **high bias**, low/no var right: low bias, **high var**	Supervised Learning _____		**High bias-Low Variance:** **Low bias-High Variance:** Trees
Validation Set	ML6.17	Require: data $\mathcal{D}$ 1: split data to disjoint subsets: training set $\mathcal{D}_{train}$, test set $\mathcal{D}_{test}$ 2: apply training only on the training set $\mathcal{D}_{train}$ 3: apply testing of learned functions on the independent test/validation set $\mathcal{D}_{test}$		Supervised Learning _____ How to spot overfitting?	+ independent set allows spotting overfitting – only subset of avail. data is used for training/testing → Cross-Validation	**Related methods:** Cross-Validation

Concept	slides	Idea	Characteristics / Parameters / Assumptions / Examples	Usage / Motivation		special cases / related methods / improvements
Cross Validation (k-fold)	ML6.x	Require: data $\mathcal{D}$, parameter k: $2 \leq k \leq N$ (N: nr of data points) 1: split data set into k disjoint subsets of equal size: $\mathcal{D}_1, \dots, \mathcal{D}_k$ 2: for $i = 1$ *to* k do 3: train model on set $\mathcal{D}_1 \cup \dots \cup \mathcal{D}_{i-1} \cup \mathcal{D}_{i+1} \cup \dots \cup \mathcal{D}_k$ (all but $\mathcal{D}_i$) 4: calculate average test error e_i on $\mathcal{D}_i$ 5: end for 6: return $\frac{1}{k}\sum_{i=1}^{k} e_i$ (average validation error)	***hyperparameter***: k: number of segments error function e_i , e.g. RMSE: $e = \sqrt{2E(w^*)/N} = \sqrt{2(y-t)^2}$ 	Supervised Learning How to spot overfitting?	+ model is learned on $\frac{k-1}{k}N$ data points and evaluated on all − model has to be learned k times	**Special Case:** leave-one-out Cross-Validation for $k = N$ if data is particularly scarce
Regularization	ML6.x	penalties (e.g. for large param values) preprocessing (remove outliers, in general filtering) more data (if errors are due to noise, increase sample size)				
Regularization Early stopping	ML6.25	stop learning when error on validation set has reached its minimum requires perpetual observation of val. error			+ −	
Regularization shrinkage methods	ML6.26	Extend error function with penalty term $E_W(w)$ $E(w) = E_D(w) + E_W(w)$ (e.g. for simple linear model $y = w^T x$) general form: sum of squares with Lq regularizer $E(w) = \frac{1}{2}\sum_{i=1}^{N}\left(t - w^T\phi(x_i)\right)^2 + \lambda\frac{1}{2}\sum_{j=1}^{M}\lvert w_j\rvert^q$	***hyperparameters***: λ: strength of regularization regularizer params (e.g. q)  case $q = 1$: L1 regularizer (Lasso: gives sparse solutions) case $q = 2$: L2 regularizer (Ridge Regression / Tikhonov reg)		+ will enforce smaller weights − determine λ (trial and error? k-fold cross val?) 	**Lasso** E_W L1 regularization: gives sparse solutions **Ridge Regression / Tikhonov regularization** E_D sum-of-squares with E_W L2 regularization $E_W(w) = \lambda\frac{1}{2}w^T w$ allows analytic sol. w = $(\lambda I + \Phi^T\Phi)^{-1}\Phi^T t$
Regularization more data	ML6.31	get more training data; if not possible directly, think about related sources of similar data				
Regularization Filtering	ML6.32	Reduce training data to a subset that is (ideally) free of noise and contains all important patterns for learning the task. Techniques: oversampling, subsampling, outlier rejection, jitering	Assume: Train data contains subset that doesn't contribute/distorts training (e.g. outliers)	unbalanced data in classification		
Regularization Feature Selection	ML6.32	feature extraction/selection: use more/less/other input features less features: reduce overfitting by avoiding pseudo relationships more/other features: improve generalization if features are related to desired output non-linear transformations:			− must be chosen problem specific	**Related methods:** (semi-)automated feature selection dimensionality reduction PCA, ICA, mutual information

Concept	slides	Idea	Characteristics / Parameters / Assumptions / Examples	Usage / Motivation		special cases / related methods / improvements
Regularization Bootstrap		Robustness towards outliers Decorrelation of the trees in the ensemble Out-of-bag error: not every data point is used to t every single tree in fact, almost 37% are not used in each tree (see assignment) predict unused points for each tree to get unbiased estimated of the generalization error			+ heavily used in practice − … “bagged” estimator has **lower variance / higher bias** than the individual models it “bags”	
Regularization Bagging (Bootstrap AGGregation)	ML6.35	train several models on bootstrap samples of the training data (data is drawn randomly with replacements) some patterns may occur twice or more, others don't occur at all average the ouput of all trained models single members of the committee might produce a higher test-set error; however in general the diversity of the committee compensates for this effect and therefore the committee error improves over the error of the individuals			+ heavily used in practice − … “bagged” estimator has **lower variance / higher bias** than the individual models it “bags”	Tree based methods
Regularization Boosting	ML6.36	ensemble technique like bagging, but models are not trained independently second model is learned on the training data that are not well learned by the first model third model is learned on the training data that are not well learned by the first and second model ... step by step, we get better commitees on the training set boosting is successfully used in practice good generalisation capabilities on low-noise data			+	
Optimization Gradient descent	ML7.25ff	Require: mathematical function f, learning rate $\epsilon > 0$ Ensure: returned vector u is close to a local minimum of f 1: choose an initial point u 2: while $\|grad\ f(u)\|$ not close to 0 do 3: $u \leftarrow u - \epsilon \cdot grad\ f(u)$ 4: end while 5: return u		Iterative Optimization for many interesting minimize problems $argmin_u\ f(u)$, no closed form solution (even though $\frac{\partial f}{\partial u_i} = 0$ holds at (local) solution points!) => iterative methods	+ … − how to choose initial u − how to choose ϵ − does this algorithm really converge?	