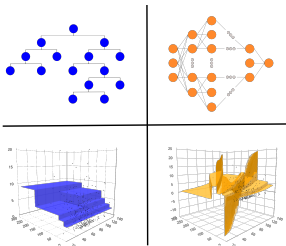
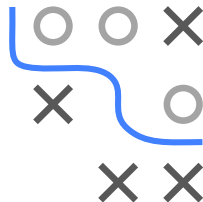


Important Learning Algorithms in ML



Learning goals

- General idea of important ML algorithms
- Overview of strengths and weaknesses

CONTENTS

- k -Nearest Neighbors (k -NN)
- Generalized Linear Models (GLM)
- Generalized Additive Models (GAM)
- Classification & Regression Trees (CART)
- Random Forests
- Gradient Boosting
- Linear Support Vector Machines (SVM)
- Nonlinear Support Vector Machines
- Neural Networks (NN)



K-NN – METHOD SUMMARY

REGRESSION

CLASSIFICATION

NONPARAMETRIC

WHITE-BOX

General idea

- **similarity** in feature space (w.r.t. certain **distance metric**
 $d(\mathbf{x}^{(i)}, \mathbf{x}) \rightsquigarrow$ similarity in target space
- **Prediction** for \mathbf{x} : construct **k -neighborhood** $N_k(\mathbf{x})$ from k points closest to \mathbf{x} in \mathcal{X} , then predict

- (weighted) mean target for **regression**: $\hat{y} = \frac{1}{\sum_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} w_i} \sum_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} w_i y^{(i)}$

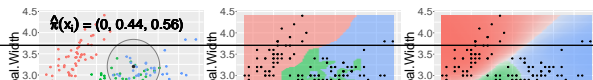
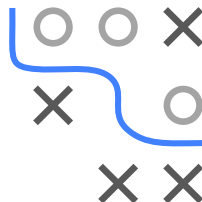
with $w_i = \frac{1}{d(\mathbf{x}^{(i)}, \mathbf{x})}$

→ optional: higher weights w_i for close neighbors

- most frequent class for **classification**: $\hat{y} = \arg \max_{\ell \in \{1, \dots, g\}} \sum_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} \mathbb{I}(y^{(i)} = \ell)$
 \Rightarrow Estimating posterior probabilities as $\hat{\pi}_\ell(\mathbf{x}^{(i)}) = \frac{1}{k} \sum_{i:\mathbf{x}^{(i)} \in N_k(\mathbf{x})} \mathbb{I}(y^{(i)} = \ell)$

- **Nonparametric** behavior: no compression of information
- Not immediately interpretable

Hyperparameters Neighborhood **size** k (locality), **distance** metric
(next page)



Classification

Left: Neighborhood for

exemplary observation in iris,

Important Learning Algorithms in ML – 2 / 28

K-NN – IMPLEMENTATION & PRACTICAL HINTS

Preprocessing Features should be standardized or normalized

Implementation

- **R:** `mlr3` learners (calling `kknn::kknn()`)
 - **Classification:**
 - `LearnerClassifKKNN`
 - `fnn::knn()`
 - **Regression:**
 - `LearnerRegrKKNN`
 - `fnn::knn.reg()`
 - Nearest Neighbour Search in $\mathcal{O}(N \log N)$: `RANN::nn2()`
- **Python:** From package `sklearn.neighbors`
 - **Classification:**
 - `KNeighborsClassifier()`
 - `RadiusNeighborsClassifier()` as alternative if data not uniformly sampled
 - **Regression:**
 - `KNeighborsRegressor()`
 - `RadiusNeighborsRegressor()` as alternative if data not uniformly sampled



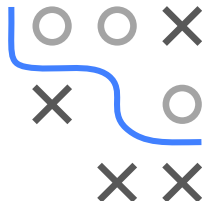
K-NN – PROS & CONS

Advantages

- + Algorithm **easy** to explain and implement
- + No distributional or functional **assumptions**
→ able to model data of **arbitrary complexity**
- + No **training** or **optimization** required
- + **local model** → **nonlinear** decision boundaries
- + Easy to **tune** (few hyperparameters)
→ number of neighbors k , distance metric
- + **Custom** distance metrics can often be easily designed to incorporate domain knowledge

Disadvantages

- Sensitivity w.r.t. **noisy** or **irrelevant** features and outliers due to dependency on distance measure
- Heavily affected by **curse of dimensionality**
- Bad performance when feature **scales** are not consistent with feature relevance
- Poor handling of data **imbalances** (worse for more global model, i.e., large k)



GENERALIZED LINEAR MODELS – METHOD SUMMARY

REGRESSION

CLASSIFICATION

PARAMETRIC

WHITE-BOX

FEATURE SELECTION

General idea Represent target as function of linear predictor $\theta^\top \mathbf{x}$ (weighted sum of features)

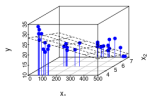
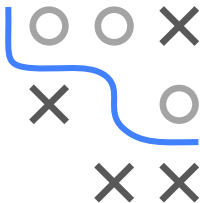
→ **Interpretation:** if feature x_j increases by 1 unit, the linear predictor changes by θ_j units

Hypothesis space $\mathcal{H} = \{f : \mathcal{X} \rightarrow \mathbb{R} \mid f(\mathbf{x}) = \phi(\theta^\top \mathbf{x})\}$, with suitable transformation $\phi(\cdot)$, e.g.,

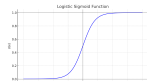
- **Linear Regression:** $\mathcal{Y} = \mathbb{R}$, ϕ identity
- **Logistic Regression:** $\mathcal{Y} = \{0, 1\}$, logistic sigmoid

$$\phi(\theta^\top \mathbf{x}) = \frac{1}{1 + \exp(-\theta^\top \mathbf{x})} =: \pi(\mathbf{x} \mid \theta)$$

⇒ Decision rule: Linear hyperplane



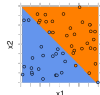
Linear regression hyperplane



Logistic sigmoid function



Logistic function for bivariate input and loss-minimal θ



Corresponding separating hyperplane

GENERALIZED LINEAR MODELS – REGULARIZATION

General idea

- Unregularized LM: risk of **overfitting** in high-dimensional space with only few observations
- **Goal**: avoidance of overfitting by adding **penalty term**

Regularized empirical risk

- Empirical risk function **plus complexity penalty** $J(\theta)$, controlled by shrinkage parameter

$\lambda > 0$:

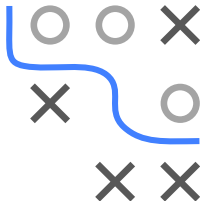
$$\mathcal{R}_{\text{reg}}(\theta) := \mathcal{R}_{\text{emp}}(\theta) + \lambda \cdot J(\theta)$$

- **Ridge** regression: L2 penalty
 $J(\theta) = \|\theta\|_2^2$
- **LASSO** regression: L1 penalty
 $J(\theta) = \|\theta\|_1$

Optimization under regularization

- **Ridge**: analytically with $\hat{\theta}_{\text{Ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$

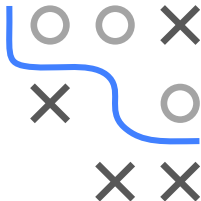
- **LASSO**: numerically with e.g. (sub-)gradient descent



GENERALIZED LINEAR MODELS – IMPLEMENTATION

Implementation

- **R:**
 - **Unregularized:** `mlr3` learner `LearnerRegrLM`, calling `stats::lm()` / `mlr3` learner `LearnerClassifLogReg`, calling `stats::glm()`
 - **Regularized / ElasticNet:** `mlr3` learners `LearnerClassifGlmnet` / `LearnerRegrGlmnet`, calling `glmnet::glmnet()`
 - For **large classification** data: `mlr3` learner `LearnerClassifLiblinearR`, calling `LiblinearR::LiblinearR()` uses fast coordinate descent
- **Python:** From package `sklearn.linear_model`
 - **Unregularized:**
 - `LinearRegression()`
 - `LogisticRegression(penalty = None)`
 - **Regularized:**
 - *Linear regression:* `Lasso()`, `Ridge()`, `ElasticNet()`
 - *Logistic regression:* `LogisticRegression(penalty =`



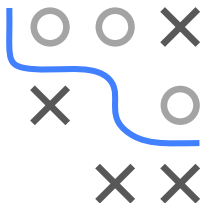
GENERALIZED LINEAR MODELS – PROS & CONS

Advantages

- + **Simple and fast** implementation
- + **Analytical** solution for L2 loss
- + Applicable for any **dataset size**, as long as number of observations \gg number of features
- + Flexibility **beyond linearity** with polynomials, trigonometric transformations, interaction terms etc.
- + Intuitive **interpretability** via feature effects
- + Statistical hypothesis **tests** for effects available

Disadvantages

- **Nonlinearity** of many real-world problems
- Further restrictive **assumptions**: linearly independent features, homoskedastic residuals, normality of conditional response
- **Sensitivity** w.r.t. outliers and noisy data (especially with L2 loss)
- Also a LM can **overfit** (e.g., many features and few observations)
- Feature **interactions** must be handcrafted



GENERALIZED ADDITIVE MODELS – METHOD SUMMARY

REGRESSION

CLASSIFICATION

(NON)PARAMETRIC

WHITE-BOX

FEATURE SELECTION

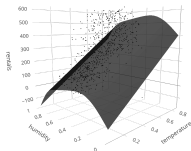
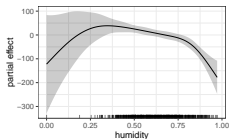


General idea

- Same as GLM, but introduce **flexibility** through **nonlinear (smooth)** effects $f_j(x_j)$
- Typically, combination of linear & smooth effects
- Smooth effects also conceivable for feature interactions

Hypothesis space $\mathcal{H} = \left\{ f : \mathcal{X} \rightarrow \mathbb{R} \mid f(\mathbf{x}) = \phi \left(\theta_0 + \sum_{j=1}^p f_j(x_j) \right) \right\},$

with suitable transformation $\phi(\cdot)$, intercept term θ_0 , and smooth functions $f_j(\cdot)$



GENERALIZED ADDITIVE MODELS – IMPLEMENTATION

Implementation

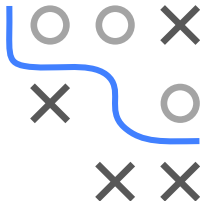
- **R:** `mlr3` learner `LearnerRegrGam`, calling `mgcv::gam()`
 - Smooth terms: `s(..., bs="<basis>")` or `te(...)` for multivariate (tensorproduct) effects
 - Link functions: `family={Gamma, Binomial, ...}`
- **Python:** `GLMGam` from package `statsmodels`; package `pygam`

Advantages

- + **Simple and fast**
- + Applicable for any **dataset size**, as long as number of observations \gg number of features
- + High **flexibility** via smooth effects
- + Easy to **combine** linear & nonlinear effects

Disadvantages

- **Sensitivity** w.r.t. outliers and noisy data
- Feature **interactions** must be handcrafted
 - practically infeasible for higher orders
- Harder to **optimize** than GLM
- Additional **hyperparameters** (type of smooth functions, smoother parameters, ...)



CART – METHOD SUMMARY

REGRESSION

CLASSIFICATION

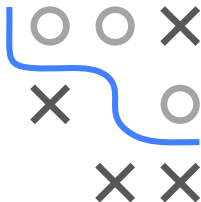
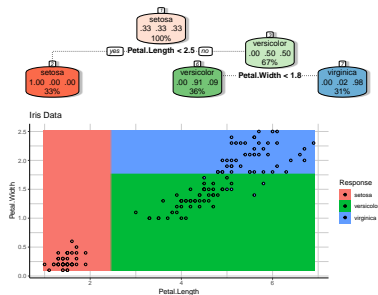
NONPARAMETRIC

WHITE-BOX

FEATURE SELECTION

General idea (CART – Classification and Regression Trees)

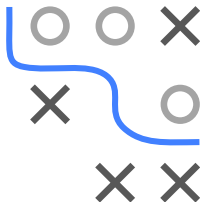
- Start at root node containing all data
- Perform repeated **axis-parallel binary splits** in feature space to obtain **rectangular partitions** at terminal nodes Q_1, \dots, Q_M
- Splits based on reduction of node **impurity**
→ empirical risk minimization (**ERM**)
- In each step:
 - Find **optimal split** (feature-threshold combination)
→ greedy search
 - Assign constant prediction c_m to all obs. in Q_m
→ Regression: c_m is average of y
→ Classif.: c_m is majority class (or class proportions)
 - Stop when a pre-defined criterion is reached
→ See **Complexity control**



CART – IMPLEMENTATION & PRACTICAL HINTS

Hyperparameters and complexity control

- Unless interrupted, splitting continues until we have pure leaf nodes (costly + overfitting)
- Hyperparameters: Complexity (i.e., number of terminal nodes) controlled via tree depth, minimum number of observations per node, maximum number of leaves, minimum risk reduction per split, ...
- Limit tree growth / complexity via
 - **Early stopping:** stop growth prematurely
→ hard to determine good stopping point before actually trying all combinations
 - **Pruning:** grow deep trees and cut back in risk-optimal manner afterwards



Implementations

- R:
 - **CART:** `mlr3` learners `LearnerClassifRpart` / `LearnerRegrRpart`, calling `rpart::rpart()`
 - **Conditional inference trees:** `partykit::ctree()`
mitigates overfitting by controlling tree size via p-value-based splitting

CART – PROS & CONS

Dual purpose of CART

- **Exploration purpose** to obtain interpretable decision rules (here: performance/tuning is secondary)
- **Prediction model**: CART as base learner in **ensembles** (bagging, random forest, boosting) can improve stability and performance (if tuned properly), but becomes less interpretable



Advantages

- + **Easy** to understand & visualize (**interpretable**)
- + Built-in **feature selection**
→ e.g., when features are not used for splitting
- + Applicable to **categorical** features
→ e.g., 2^m possible binary splits for m categories
→ trick for regr. with L2-loss and binary classif.: categories can be sorted $\Rightarrow m - 1$ binary splits

Disadvantages

- Rather **poor generalization**
- High **variance/instability**: model can change a lot when training data is minimally changed
- Can **overfit** if tree is grown too deep
- Not well-suited to model **linear** relationships
- **Bias** toward features with many unique values or categories

- + Handling of **missings** possible via surrogate splits

RANDOM FORESTS – METHOD SUMMARY

REGRESSION

CLASSIFICATION

NONPARAMETRIC

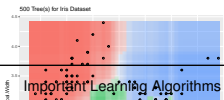
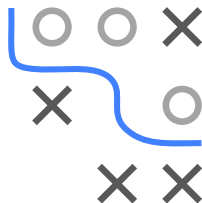
BLACK-BOX

FEATURE SELECTION

General idea

- **Bagging ensemble** of M tree **base learners** fitted on **bootstrap** data samples
 - ⇒ Reduce **variance** by ensembling while slightly increasing **bias** by bootstrapping
 - Use unstable, **high-variance** base learners by letting trees grow to full size
 - Promoting **decorrelation** by random subset of candidate features for each split
- **Predict** via averaging (regression) or majority vote (classification) of base learners

Hypothesis space $\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M \sum_{t=1}^{T^{[m]}} c_t^{[m]} \mathbb{I}(\mathbf{x} \in Q_t^{[m]}) \right\}$



RANDOM FORESTS – IMPLEMENTATION & PRACTICAL HINTS

Extremely Randomized Trees

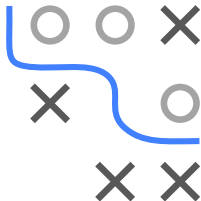
- Variance of trees can be further increased by **randomizing split points** instead of using the optimal one
- Alternatively consider k random splits and pick the best one according to impurity

Tuning

- **Ensemble size** should not be tuned as it only decreases variance
—→ choose sufficiently large ensemble
- While default values for **number of split points** is often good, tuning it can still improve performance
- Tuning the **minimum samples in leafs** and **minimum samples for splitting** can be beneficial but no huge performance increases are to be expected

Implementation

- **R:** `mlr3` learners `LearnerClassifRanger /`
`LearnerRegrRanger`, calling `ranger::ranger()` as a highly
efficient and flexible implementation



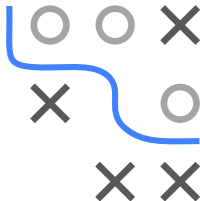
RANDOM FORESTS – PROS & CONS

Advantages

- + Retains most of **trees'** advantages (e.g., feature selection, feature interactions)
- + Fairly **good predictor**: mitigating base learners' variance through bagging
- + Quite **robust** w.r.t. small changes in data
- + Good with **high-dimensional** data, even in presence of noisy features
- + Easy to **parallelize**
- + Robust to its hyperparameter configuration
- + Intuitive measures of **feature importance**

Disadvantages

- Loss of individual trees' **interpretability**
- Can be suboptimal for **regression** when extrapolation is needed
- **Bias** toward selecting features with many categories (same as CART)
- Rather large model size and slow inference time for large ensembles
- Typically inferior in **performance** to tuned gradient tree boosting.



GRADIENT BOOSTING – METHOD SUMMARY

REGRESSION

CLASSIFICATION

(NON)PARAMETRIC

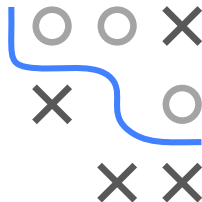
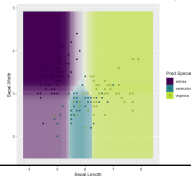
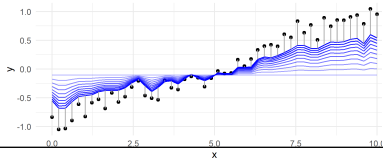
BLACK-BOX

FEATURE SELECTION

General idea

- **Sequential ensemble** of M **base learners** by greedy forward stagewise additive modeling
 - In each iteration a base learner is fitted to current **pseudo residuals** \Rightarrow one boosting iteration is one approximate **gradient step in function space**
 - Base learners are typically **trees**, **linear regressions** or **splines**
- **Predict** via (weighted) sum of base learners

Hypothesis space $\mathcal{H} = \left\{ f(\mathbf{x}) : f(\mathbf{x}) = \sum_{m=1}^M \beta_m b(\mathbf{x}, \theta_m) \right\}$



GRADIENT BOOSTING – PRACTICAL HINTS

- **Feature and data subsampling** for each base learner fit
- **Parallelization** and **approximate split finding** for tree base learners
- GPU acceleration

Explainable / Componentwise Gradient Boosting

Tuning

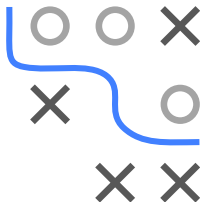
GRADIENT BOOSTING – PROS & CONS

Advantages

- + Retains most of **base learners'** advantages
- + Very **good predictor** due to aggressive loss minimization, typically only outperformed by heterogenous **stacking ensembles**
- + High **flexibility** via custom loss functions and choice of base learner
- + Highly efficient implementations exist (`lightgbm` / `xgboost`) that work well on large (distributed) data sets
- + Componentwise boosting: Good combination of (a) high performance (b) interpretable model and (c) feature selection

Disadvantages

- Loss of base learners' potential **interpretability**
- **Many hyperparameters** to be carefully tuned
- Hard to **parallelize** (↪ solved by efficient implementation)



LINEAR SVM – METHOD SUMMARY

CLASSIFICATION

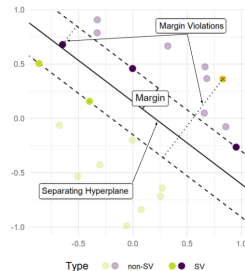
REGRESSION

PARAMETRIC

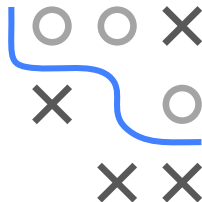
WHITE-BOX

General idea (Soft-margin SVM)

- Find linear decision boundary (**separating hyperplane**) that
 - maximizes distance (**margin γ**) to closest points (**support vectors, SVs**) on each side of decision boundary
 - while minimizing margin violations (points either on **wrong side of hyperplane** or **between dashed margin line and hyperplane**)
- 3 types of training points
 - non-SVs** with no impact on decision boundary
 - SVs that are margin violators** and affect decision boundary
 - SVs located exactly on dashed margin lines** and affect decision boundary



Soft-margin SVM with margin violations

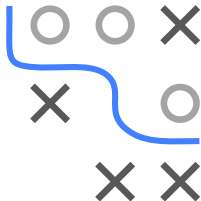
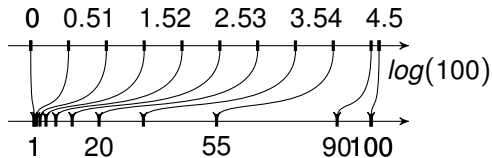


Hypothesis space (primal) $\mathcal{H} = \{f(\mathbf{x}) : f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + \theta_0\}$

Preprocessing Features should be scaled before applying SVMs
(applies generally to regularized models)

Tuning

- Tuning of cost parameter C advisable
⇒ strong influence on resulting hyperplane
- C it is often tuned on a log-scale grid for optimal and space-filling search space



Implementation

- **R:** `mlr3` learners `LearnerClassifSVM` / `LearnerRegrSVM`, calling `e1071::svm()` with linear kernel (`libSVM` interface). Further implementations in `mlr3extralearners` based on
 - `kernlab::ksvm()` allowing custom kernels
 - `LiblinearR::LiblinearR()` for a fast implementation with linear kernel

- **Python:** `sklearn.svm.SVC` from package `scikit-learn` /

package `libSVM`

NONLINEAR SVM – METHOD SUMMARY

CLASSIFICATION

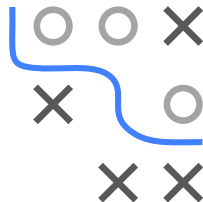
REGRESSION

NONPARAMETRIC

BLACK-BOX

General idea

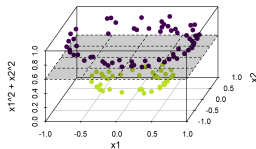
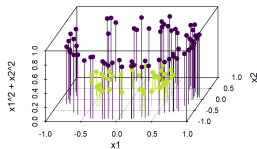
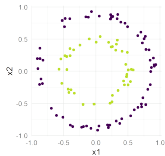
- Move **beyond linearity** by mapping data to transformed space where they are linearly separable
- **Kernel trick**
 - No need for explicit construction of feature maps
 - Replace inner product of feature map $\phi : \mathcal{X} \rightarrow \Phi$ by **kernel**:
 $\langle \phi \mathbf{x}, \phi \mathbf{x}^t \rangle = k(\mathbf{x}, \mathbf{x}^t)$



Hypothesis space

$$\mathcal{H} = \{f(\mathbf{x}) : f(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^\top \phi \mathbf{x} + \theta_0)\} \text{ (primal)}$$

$$\mathcal{H} = \left\{f(\mathbf{x}) : f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^n \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0\right) \mid \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y^{(i)} = 0\right\} \text{ (dual)}$$



Nonlinear problem in original space

Mapping to 3D space and subsequent linear separation – implicitly handled by kernel in nonlinear SVM

Dual problem **Kernelize** dual (soft-margin) SVM problem, replacing all inner

products by kernels:

NONLINEAR SVM – IMPLEMENTATION & PRACTICAL HINTS

- **Linear** kernel: dot product of given observations $\Rightarrow k_{\text{xx}t} = \mathbf{x}^\top \tilde{\mathbf{x}} \Rightarrow$ linear SVM
- **Polynomial** kernel of degree $d \in \mathbb{N}$: monomials (i.e., feature interactions) up to d -th order $\Rightarrow k_{\text{xx}t} = (\mathbf{x}^\top \tilde{\mathbf{x}} + b)^d, b \geq 0$
- **Radial basis function (RBF)** kernel: infinite-dimensional feature space, allowing for perfect separation of all finite datasets $\Rightarrow k_{\text{xx}t} = \exp(-\gamma \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2)$ with bandwidth parameter $\gamma > 0$

Tuning

Implementation

SVM – PRO'S & CON'S

Advantages

- + Often **sparse** solution (w.r.t. observations)
- + Robust against overfitting (**regularized**); especially in high-dimensional space
- + **Stable** solutions (w.r.t. changes in train data)
 - Non-SV do not affect decision boundary
- + Convex optimization problem
 - local minimum $\hat{=}$ global minimum

Advantages (nonlinear SVM)

- + Can learn **nonlinear decision boundaries**
- + **Very flexible** due to custom kernels
 - RBF kernel yields local model

Disadvantages

- **Long** training times $\rightarrow O(n^2 p + n^3)$
- Confined to **linear model**
- Restricted to **continuous features**
- Optimization can also fail or get stuck



Disadvantages (nonlinear SVM)

- Poor **interpretability** due to complex kernel
- **Not easy tunable** as it is highly important to choose the right kernel (which also

NEURAL NETWORKS – METHOD SUMMARY

REGRESSION

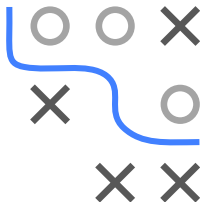
CLASSIFICATION

(NON)PARAMETRIC

BLACK-BOX

General idea

- Learn **composite function** through series of nonlinear feature transformations, represented as **neurons**, organized hierarchically in **layers**
 - Basic neuron operation: 1) affine **transformation** ϕ (weighted sum of inputs), 2) nonlinear **activation** σ
 - Combinations of simple building blocks to create a complex model
- Optimize via **mini-batch stochastic gradient descent (SGD)**
variants:
 - Gradient of each weight can be inferred from the **computational graph** of the network
→ **Automatic Differentiation** (AutoDiff)
 - Algorithm to compute weight updates based on the loss is called **Backpropagation**



Hypothesis space $\mathcal{H} =$

$$\{f(\mathbf{x}) : f(\mathbf{x}) = \tau \circ \phi \circ \sigma^{(h)} \circ \phi^{(h)} \circ \sigma^{(h-1)} \circ \phi^{(h-1)} \circ \sigma^{(1)} \circ \phi^{(1)}(\mathbf{x})\}$$

NEURAL NETWORKS – IMPLEMENTATION & PRACTICAL HINTS

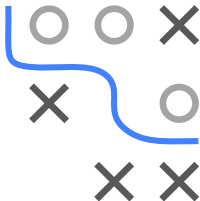
General hints

- Instead of NAS, use a standard architecture and tune training hyperparameters
- Training pipeline (data-augmentation, training schedules, ...) is more crucial than the specific architecture
- While NNets are state-of-the-art for **computer vision (CV)** and **natural language processing (NLP)**, we recommend not to use them for tabular data because alternatives perform better
- Computational efforts for training (and inference) can be very high, requiring specific hardware.
→ Using a service (esp. for foundation models) can be more cost efficient

Implementation

- **R:** Use python libraries (below) via `reticulate`, but not really recommended except for toy applications.
- **Python libraries:**

- `keras` for simple high level API



NEURAL NETWORKS – PROS & CONS

Advantages

- + Applicable to **complex, nonlinear** problems
- + Very **versatile** w.r.t. architectures
- + State-of-the-art for CV and NLP
- + Strong **performance** if done right
- + Built-in **feature extraction**, obtained by intermediate representations
- + Easy handling of **high-dimensional** data
- + **Parallelizable** training

Disadvantages

- Typically, high computational **cost**
- High demand for **training data**
- Strong tendency to **overfit**
- Requiring lots of **tuning expertise**
- **Black-box** model – hard to interpret or explain

