Common Machine Learning Algorithms
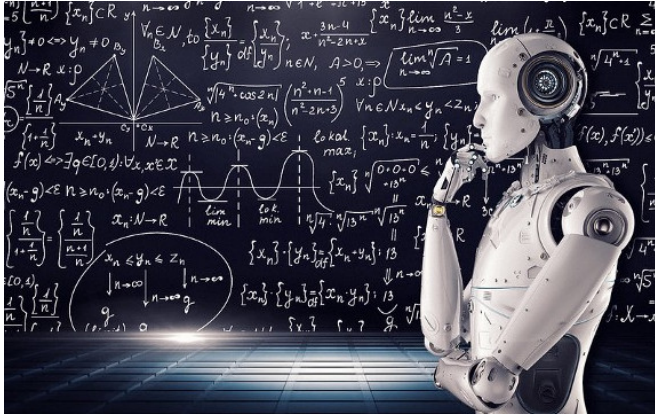
CONTENTS

LINEAR MODELS

LINEAR MODELS – FUNCTIONALITY

General idea   Represent target as function of linear predictor $\boldsymbol{\theta}^T \mathbf{x}$

Hypothesis space

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

- Identity $\phi(\boldsymbol{\theta}^\top \mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x}$ $\Rightarrow$ **linear regression**

- Logistic sigmoid function $\phi(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^T \mathbf{x})} =: \pi(\mathbf{x} \mid \boldsymbol{\theta})$ $\Rightarrow$ **(binary) logistic regression**

  - Probability $\pi(\mathbf{x} \mid \boldsymbol{\theta}) = \mathbb{P}(y = 1 \mid \mathbf{x})$ of belonging to one of two classes
  - Separating hyperplane via decision rule (e.g., $\hat{y} = 1 \Leftrightarrow \pi(\mathbf{x}) > 0.5$)
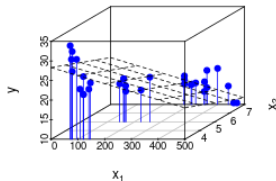
Linear regression hyperplane   Logistic function for bivariate input and loss-minimal $\boldsymbol{\theta}$ Corresponding separating hyperplane
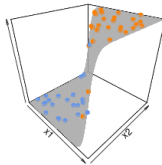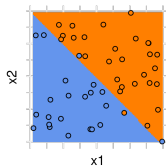
LINEAR MODELS – FUNCTIONALITY

Empirical risk

- **Linear regression**

  - Typically, based on **quadratic** loss: $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum\limits_{i=1}^{n} \left(y^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) \right)^2$

    $\Rightarrow$ corresponding to ordinary-least-squares (OLS) estimation
  - Alternatives: e.g., **absolute** or **Huber** loss (both improving robustness)

- **Logistic regression:** based on **Bernoulli/log/cross-entropy** loss

  $\Rightarrow \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum\limits_{i=1}^{n} -y^{(i)} \log \left(\pi \left(\mathbf{x}^{(i)} \right) \right) - (1 - y^{(i)}) \log \left(1 - \pi \left(\mathbf{x}^{(i)} \right) \right)$

Optimization

- For **OLS**: analytically with $\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$
  (with $\mathbf{X} \in \mathbb{R}^{n \times p}$: matrix of feature vectors)

- For **other loss functions**: numerical optimization

Hyperparameters   None

LINEAR MODELS – PRO'S & CON'S

Advantages

- $+$ **Simple and fast** implementation
- $+$ **Analytical** solution
- $+$ **Cheap** computation
- $+$ Applicable for any **dataset size**, as long as number of observations $\gg$ number of features
- $+$ Flexibility **beyond linearity** with polynomials, trigonometric transformations 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)
- $-$ Risk of **overfitting** in higher dimensions
- $-$ Reature **interactions** must be handcrafted, so higher orders practically infeasible
- $-$ No handling of **missing** data

Simple, highly interpretable method suited for linear problems, but with strong assumptions, practical limitations, and tendency to overfit

LINEAR MODELS – REGULARIZATION

Idea

- Unregularized LM: risk of **overfitting** in high-dimensional space with only few observations
- **Goal**: find compromise between model fit and generalization by adding **penalty term**

Regularized empirical risk

- Empirical risk function **plus complexity penalty** $J(\boldsymbol{\theta})$, controlled by shrinkage parameter $\lambda > 0$:
  $$\mathcal{R}_{\text{reg}}(\boldsymbol{\theta}) := \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) + \lambda \cdot J(\boldsymbol{\theta}).$$
- Popular regularizers
    - **Ridge** regression: L2 penalty $J(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$
    - **LASSO** regression: L1 penalty $J(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$

Optimization under regularization

- **Ridge**: analytically with $\hat{\boldsymbol{\theta}}_{\text{Ridge}} = (\mathbf{X}^\top \mathbf{X} + \lambda \boldsymbol{I})^{-1} \mathbf{X}^\top \mathbf{y}$
- **LASSO**: numerically with, e.g., (sub-)gradient descent

LINEAR MODELS – REGULARIZATION

Choice of regularization parameter

- Standard hyperparameter optimization problem
- E.g., choose λ with minimum mean cross-validated error (default in R package `glmnet`)

Ridge vs. LASSO

- **Ridge**
    - Overall smaller, but still dense θ
    - Suitable with many influential features present, handling correlated features by shrinking their coefficients equally

- **LASSO**
    - Actual variable selection
    - Suitable for sparse problems, ineffective with correlated features (randomly selecting one)

- Neither overall better – compromise: **elastic net**
  $\rightarrow$ weighted combination of Ridge and LASSO regularizers

LINEAR MODELS – PRACTICAL HINTS

Implementation

- **R:**
  - **Unregularized:** mlr3 learner `LearnerRegrLM`, calling `stats::lm()` / mlr3 learner `LearnerClassifLogReg`, calling `stats::glm()`
  - **Regularized:** mlr3 learners `LearnerClassifGlmnet` / `LearnerRegrGlmnet`, calling `glmnet::glmnet()`
- **Python:** `LinearRegression` from package `sklearn.linear_model`, package for advanced statistical parameters `statsmodels.api`

LINEAR SUPPORT VECTOR MACHINES

LINEAR SVM – FUNCTIONALITY

SUPERVISED   CLASSIFICATION   PARAMETRIC   BLACK-BOX

General idea

- Find linear decision boundary (**separating hyperplane**) that best separates classes
    - **Hard-margin** SVM: maximize distance (**margin** $\gamma > 0$) to closest members (**support vectors, SV**) on each side of decision boundary
    - **Soft-margin** SVM: relax separation to allowing margin violations $\rightarrow$ maximize margin while minimizing violations
- 3 types of training points
    - **non-SVs** with no impact on decision boundary
    - **SVs** located exactly on decision boundary
    - **margin violators**

Hypothesis space $\mathcal{H} = \left\{ f : \mathcal{X} \rightarrow \mathbb{R} \mid f(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} + \theta_0 \right\}$ separater intercept notwendig?

LINEAR SVM – FUNCTIONALITY
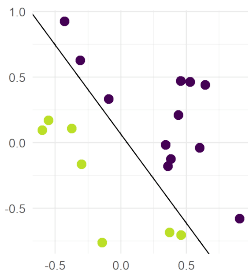
Hard-margin SVM: margin is maximized by boundary on the right
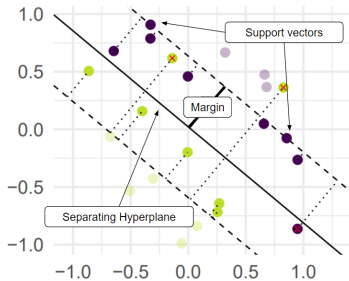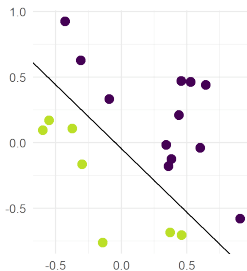
Soft-margin SVM with margin violations

Dual problem

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \left\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right\rangle$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \ \ \forall i \in \{1, \ldots, n\} \ \ (C = \infty \text{ for hard-margin SVM}),$$

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

LINEAR SVM – FUNCTIONALITY

Empirical risk

Soft-margin SVM also interpretable as **L2-regularized ERM**:

$$\frac{1}{2}\|\boldsymbol{\theta}\|^2 + C \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$

with

- $\|\boldsymbol{\theta}\| = 1/\gamma$,
- $C > 0$: penalization for missclassified data points
- $L(y, f) = \max(1 - yf, 0)$: **hinge** loss
  $\rightarrow$ other loss functions applicable (e.g., **Huber** loss)

Optimization

- Typically, tackling **dual** problem (though feasible in corresponding primal) via **quadratic programming**
- Popular: **sequential minimal optimization** $\rightarrow$ iterative algorithm based on breaking down objective into bivariate quadratic problems with analytical solutions

Hyperparameters Cost parameter C

LINEAR SVM – PRO'S & CON'S

Advantages

+ Often **sparse** solution
+ Robust against overfitting (**regularized**); especially in high-dimensional space
+ **Stable** solutions, as non-SV do not influence decision boundary

Disadvantages

— **Costly** implementation; long training times
— **Limited scalability** to larger data sets **??**
— Confined to **linear separation**
— Poor **interpretability**

Very accurate solution for high-dimensional data that is linearly separable

LINEAR SVM – PRACTICAL HINTS

Preprocessing

Features must be rescaled before applying SVMs.

Tuning

Cost parameter *C* must be tuned and has strong influence on resulting separating hyperplane.

Implementation

- **R:** mlr3 learners `LearnerClassifSVM` / `LearnerRegrSVM`, calling `svm()` from `libsvm`
- **Python:** `sklearn.svm.SVC` from package `scikit-learn` / package `libSVM`

NONLINEAR SUPPORT VECTOR MACHINES

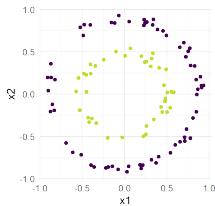# NONLINEAR SVM – FUNCTIONALITY

SUPERVISED CLASSIFICATION NON-PARAMETRIC BLACK-BOX

General idea

- Move **beyond linearity** by mapping data to transformed space where they are linearly separable
- **Kernel trick** (based on Mercer's theorem, existende of RKHS):
  - Replace two-step operation feature map $\phi \rightsquigarrow$ inner product by **kernel** $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, s.t. $\langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}), = \rangle k(\mathbf{x}, \tilde{\mathbf{x}})$
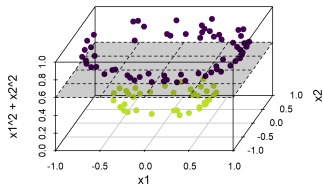 - No need for explicit construction of feature maps; very fast and flexible

Hypothesis space $\quad \mathcal{H} = \left\{ f(\mathbf{x}) = \sum\limits_{i=1}^{n} \alpha_i y^{(i)} k(\mathbf{x}^{(i)}, \mathbf{x}) + \theta_0 | \ \theta_0, \alpha_i \in \mathbb{R} \right\}$

Data are not linearly separable in original space.
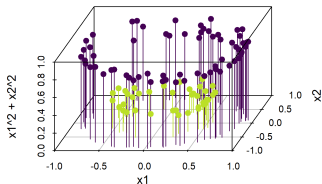
Mapping to 3D space allows for linear separation with hyperplane.

NONLINEAR SVM – FUNCTIONALITY

Dual problem

Kernelize dual (soft-margin) SVM problem, replacing all inner products $\left\langle \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \right\rangle$ by kernels:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}), \ \ \text{s.t.} \ \ 0 \leq \alpha_i \leq C, \ \ \sum_{i=1}^{n} \alpha_i y^{(i)} = 0.$$
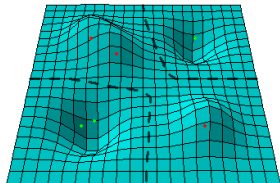
Hyperparameters Cost C of margin violations, kernel hyperparameters (e.g., width of RBF kernel)

Interpretation as basis function approach

- **Representer theorem:** dual soft-margin SVM problem can be expressed through
  $$\boldsymbol{\theta} = \sum_{j=1}^{n} \beta_j \phi\left(\mathbf{x}^{(j)}\right)$$
- Sparse, weighted sum of **basis functions** with $\beta_j = 0$ for non-SVs
- **Local** model, with smoothness depending on kernel properties

RBF kernel as mixture of Gaussian basis functions, forming bumpy, nonlinear decision surface to discern red and green points.

NONLINEAR SVM – PRO'S & CON'S

Advantages

- $+$ high **accuaracy**
- $+$ can learn **nonlinear decision boundaries**
- $+$ often **sparse** solution
- $+$ robust against overfitting (**regularized**); especially in high-dimensional space
- $+$ **stable** solutions, as the non-SV do not influence the separating hyperplane

Disadvantages

- $-$ **costly implementation**; long training times
- $-$ does not scale well to **larger data sets ??**
- $-$ only **linear separation** $\rightarrow$ possible with nonlinear SVMs which are explained in the following slides.
- $-$ poor **interpretability**
- $-$ **not easy tunable** as it is highly important to choose the right kernel

nonlinear SVMs perform very well for nonlinear separable data, but are hard to interpret and need a lot of tuning.

NONLINEAR SVM – PRACTICAL HINTS

Kernels

Mainly, these three types of kernels are used:

- **Linear kernel**: the dot product of the given observations
- **Polynomial kernel**: curved lines in the input space
- **Radial basis function (RBF) kernel**: complex regions in the input space (e. g. spirals)

Tuning

- SVMs are sensitive to its hyperparameters and **should always be tuned**.
- For the RBF kernel the **RBF sigma heuristic** is used.

Implementation

- **R:** mlr3 learners LearnerClassifSVM / LearnerRegrSVM, calling e1071::svm() (interface to libSVM)
- **Python:** sklearn.svm.SVC from package scikit-learn / package libSVM