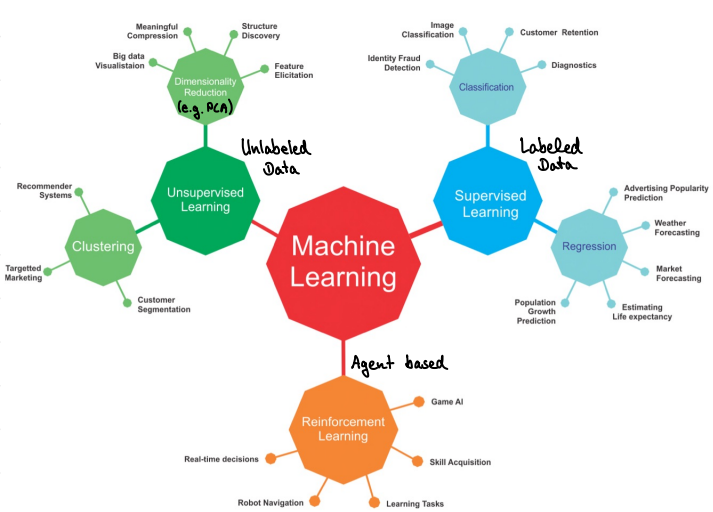
Introduction to Machine Learning

I2ML (oder MaLe)

Legende

- Neus Kapitel
- Unterkapitel
- Definition
- Idee/Setup/Motivation für den darauffolgenden Inhalt
- Wichtiger Teil einer Definition/Aussage/Theorem
- Notation
- Beispiele

Machine Learning Basics



Data in Supervised Learning

	Features x				Target y
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
labeled data	4.3	3.0	1.1	0.1	setosa
	5.0	3.3	1.4	0.2	setosa
	7.7	3.8	6.7	2.2	virginica
	5.5	2.5	4.0	1.3	versicolor
unlabeled data	5.9	3.0	5.1	1.8	?
	4.4	3.2	1.3	0.2	?

labels

n-Zeilen

X_2

p-Merkmale

Target variable types:

- Numerical ($\mathbb{R}$)
- Integer ($\mathbb{Z}$)
- Categorical ($\{C_1, \dots, C_g\}$) } Regression task
- Binary ($\{0,1\}$) } Classification task

} Supervised task

Models & Parameters

In most Regression tasks $g = 1$. In Classification tasks $g = \#$ Categories and f is e.g. a score.

A function $f: \mathcal{X} \to \mathbb{R}^g$ is called a **Model** (or **Hypothesis**)

! ML requires contraining f to a certain type of function which we call **structual prior**.

The set $\mathcal{H} := \{f \mid f \text{ belongs to the class of the structual prior}\}$ is called **Hypothesis space / Model class**.

Usually $\mathcal{H}$ is constructed as a parametrized family of curves with parameters $\underline{\theta} := (\theta_1, \dots, \theta_d) \in \Theta$ **Parameter space**

$\Rightarrow \mathcal{H} = \{f_\theta \mid f_\theta \text{ belongs to a family of curves parametrized by } \theta\}$

Therefore: ! Finding the optimal model is equivalent to finding the optimal parameters

! The parameter-to-model mapping can be non-injective, i.e. one model is described by different parameter vectors.

Notation

Input/Feature Space: $\mathcal{X}$ } All data sets of size n: $\mathbb{D}_n := (\mathcal{X} \times \mathcal{Y})^n$

Output/Target Space: $\mathcal{Y}$ } All finite data sets: $\mathbb{D} := \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n$

For the actually observed data we denote:

i-th observation: $(\underline{x}^{(i)}, y^{(i)}) \in \mathcal{X} \times \mathcal{Y}$

j-th feature vector: $\underline{x}_j := (x_j^{(1)}, \dots, x_j^{(n)})^T$

Observed data set: $\mathcal{D} := \left((\underline{x}^{(1)}, y^{(1)}), \dots, (\underline{x}^{(n)}, y^{(n)})\right) \in \mathbb{D}_n$

Data generating process: $\mathbb{P}_{xy}: \mathcal{X} \times \mathcal{Y} \to [0,1]$

Encoding of Categorical Features

Let x be a **nominal-scaled** feature, i.e. $x \in \{C_1, \dots, C_k\}$

We transform the scalar x into a vector: $o(x) := \begin{bmatrix} \mathbb{1}_{\{C_1\}}(x) \\ \vdots \\ \mathbb{1}_{\{C_{\tilde{k}}\}}(x) \end{bmatrix} \in \{0,1\}^{\tilde{k}}$.

Each entry of $o(x)$ is treated as a seperate (binary) feature.

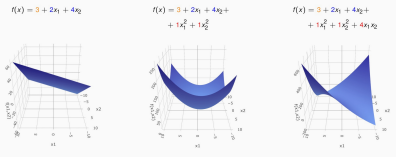
- **One-hot-encoding**: $\tilde{k} = k$ dummies. So exactly one entry of $o(x)$ is 1 ("hot").
- **Dummy-encoding**: $\tilde{k} = k-1$ dummies. So at most one entry of $o(x)$ is 1.
  - ↳ cuts redundancy of one-hot-encoding
  - Necessary if non-singular-matrix is required (e.g. Lin.Reg.)

For an **ordinal-scaled** feature x we use an encoding that reflects the ordinality. e.g. a sequence of integer values.

Example for Hypothesis/Parameter Spaces

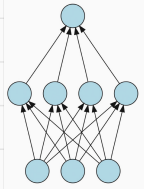
Bivariate quadratic function:

$$\mathcal{H} = \{f : f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2, \theta \in \mathbb{R}^6\},$$



RBF Network.

$$\mathcal{H} = \left\{f : f(\mathbf{x}) = \sum_{i=1}^{k} a_i \rho(\|\mathbf{x} - \mathbf{c}_i\|)\right\},$$



a_i := weight of i-th neuron

c_i := its center vector

$\rho(\|\vec{x} - c_i\|) := \exp(-\beta \cdot \|\vec{x} - c_i\|^2)$ is the i-th radial basis function with bandwidth $\beta \in \mathbb{R}$.

Losses & Risk Minimization

Loss

The Loss function $L : Y \times \mathbb{R}^g \longrightarrow \mathbb{R}$, $(y, f(x)) \mapsto L(y, f(x))$ quantifies the quality of the prediction on a single observation x.

Risk

Given a model $f(x)$ and a Loss function L, the (theoretical) Risk is the expected loss

$$\mathcal{R}(f) := \mathbb{E}_{xy}[L(y, f(x))] = \int L(y, f(x)) \, d\mathbb{P}_{xy}$$

Empirical Risk

Given a model $f(x)$, a Loss function L and an i.i.d. sample $x^{(1)}, \ldots, x^{(n)}$ from $\mathbb{P}_{xy}$, then the Empirical Risk is defined as

$$\mathcal{R}_{emp}(f) := \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)})) \text{ or (parametrized) } \mathcal{R}_{emp}(\theta) := \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)} | \theta))$$

Goal

The best model is the model with the smallest risk. We estimate that by computing $\hat{\theta} := \arg\min_{\theta \in \Theta} \mathcal{R}_{emp}(\theta)$

Optimization

Empirical Risk Minimization

The ERM problem is $\hat{\theta} := \arg\min_{\theta \in \Theta} \mathcal{R}_{emp}(\theta)$

! $\hat{\theta}$ doesn't need to be unique.

Example

- Gradient Descent (if $\mathcal{R}_{emp}$ is continuous in θ).
- Normal-equation in case of linear regression

Components of Supervised Learning

Learner/Inducer

A Learner/Inducer I is an algorithm which

- receives a training set $D \in \mathbb{D}$ and
- for a given hypothesis space $\mathcal{H}$ of models $f : \mathcal{X} \to \mathbb{R}^g$
- uses a risk function $\mathcal{R}_{emp}(f)$ or $\mathcal{R}_{emp}(\theta)$ to evaluate $f \in \mathcal{H}$ on D.
- uses an optimization procedure to find $\hat{f}$ or $\hat{\theta}$

So the inducer is a mapping $I : \mathbb{D} \times \Lambda \to \mathcal{H}$ or $I : \mathbb{D} \times \Lambda \to \Theta$

(Λ: Hyperparameterspace for control settings λ)

Framework

Learning = $\mathcal{H}$ + Risk + Optimization

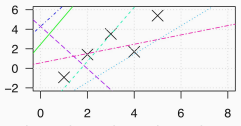
= $\mathcal{H}$ + Loss(+ Regularization) + Optimization

Hypothesis Space	Risk / Loss	Optimization
Step functions	Mean squared error	Analytical solution
Linear functions	Misclassification rate	Gradient descent
Sets of rules	Negative log-likelihood	Combinatorial optimization
Neural networks	Information gain	Genetic algorithms
Voronoi tesselatio	...	...
...		

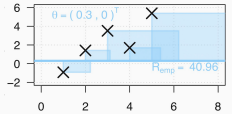
! Useful framework, but doesn't cover all special cases. E.g. some ML methods are not defined via risk minimization.

Example

$\mathcal{H} = \{f(\mathbf{x}) = \theta_0 + \theta \mathbf{x} : \theta_0, \theta \in \mathbb{R}\}$ $\quad \mathcal{R}_{emp}(\theta) = \sum_{i=1}^{n} (y^{(i)} - \theta_0 - \theta x^{(i)})^2$

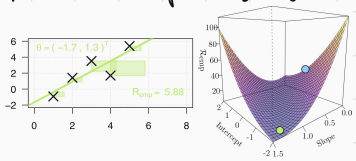


Alt: Polynomials, Splines



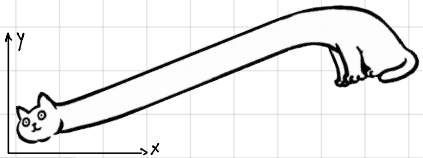
Alt: Absolute error, L1 Loss-function

Opt. method: Deriving $\hat{\theta}$ analytically.



Alt: Stochastic gradient descent

Supervised Regression



Design matrix

Let $\underline{x}_1, \dots, \underline{x}_p$ be our p feature vectors, then we call the matrix

$X = \begin{pmatrix} 1 & | & | & & | \\ \vdots & \underline{x}_1 & \underline{x}_2 & \dots & \underline{x}_p \\ 1 & | & | & & | \end{pmatrix} \in \mathbb{R}^{n \times (p+1)}$ the design matrix.

Furthermore, the define $\underline{x} := (1, x_1, \dots, x_p)^T \in \mathbb{R}^{p+1}$ and $\underline{\theta} = (\theta_0, \theta_1, \dots, \theta_p) \in \mathbb{R}^{p+1}$

Setup

We predict $y \in \mathbb{R}$ as linear combinations of features

$\hat{y} = f(\underline{x}) = \underline{\theta}^T \underline{x} = \theta_0 + \theta_1 x_1 + \dots + \theta_p x_p$

This results in the hypothesis space $\mathcal{H} = \{ f(\underline{x}) = \underline{\theta}^T \underline{x} \mid \underline{\theta} \in \mathbb{R}^{p+1} \}$

A typical choice for the Loss-function is L2-Loss.

This results in $\mathcal{R}_{emp}(\underline{\theta}) = \sum_{i=1}^{n} (y^{(i)} - \underline{\theta}^T \underline{x}^{(i)})^2$.

Optimization

We want to find $\hat{\underline{\theta}} = \arg\min_{\theta \in \Theta} \sum_{i=1}^{n} (y^{(i)} - \underline{\theta}^T \underline{x}^{(i)})^2 = \arg\min_{\theta \in \Theta} \| \underline{y} - X\underline{\theta} \|_2^2$

This results in the ordinary-least-squares estimator:

$\hat{\underline{\theta}} = (X^T X)^{-1} X^T \underline{y}$ (MLE)

ANALYTICAL OPTIMIZATION – PROOF

$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^{n} (\underbrace{y^{(i)} - \theta^\top \mathbf{x}^{(i)}}_{=:\epsilon_i})^2 = \| \underbrace{\mathbf{y} - \mathbf{X}\theta}_{=:\epsilon} \|_2^2; \quad \theta \in \mathbb{R}^{\tilde{p}} \text{ with } \tilde{p} := p + 1$

$$
\begin{aligned}
0 &= \frac{\partial \mathcal{R}_{\text{emp}}(\theta)}{\partial \theta} && \text{(sum notation)} \\
0 &= \frac{\partial}{\partial \theta} \sum_{i=1}^{n} \epsilon_i^2 \quad | \text{ sum \& chain rule} \\
0 &= \sum_{i=1}^{n} \frac{\partial \epsilon_i^2}{\partial \epsilon_i} \frac{\partial \epsilon_i}{\partial \theta} \\
0 &= \sum_{i=1}^{n} 2\epsilon_i (-1) (\mathbf{x}^{(i)})^\top \\
0 &= \sum_{i=1}^{n} (y^{(i)} - \theta^\top \mathbf{x}^{(i)}) (\mathbf{x}^{(i)})^\top \\
\sum_{i=1}^{n} \theta^\top \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^\top &= \sum_{i=1}^{n} y^{(i)} (\mathbf{x}^{(i)})^\top \quad | \text{ transpose} \\
\theta \sum_{i=1}^{n} \underbrace{(\mathbf{x}^{(i)})^\top \mathbf{x}^{(i)}}_{1 \times 1} &= \sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \\
\theta &= \sum_{i=1}^{n} \underbrace{(\underbrace{(\mathbf{x}^{(i)})^\top \mathbf{x}^{(i)}}_{1 \times 1})^{-1}}_{\tilde{p} \times 1} \underbrace{\mathbf{x}^{(i)}}_{\tilde{p} \times 1} \underbrace{y^{(i)}}_{1 \times 1}
\end{aligned}
$$

$$
\begin{aligned}
0 &= \frac{\partial \mathcal{R}_{\text{emp}}(\theta)}{\partial \theta} && \text{(matrix notation)} \\
0 &= \frac{\partial \|\epsilon\|_2^2}{\partial \theta} \\
0 &= \frac{\partial \epsilon^\top \epsilon}{\partial \theta} \quad | \text{ chain rule} \\
0 &= \frac{\partial \epsilon^\top \epsilon}{\partial \epsilon} \cdot \frac{\partial \epsilon}{\partial \theta} \\
0 &= 2\epsilon^\top \cdot (-1 \cdot \mathbf{X}) \\
0 &= (\mathbf{y} - \mathbf{X}\theta)^\top \mathbf{X} \\
0 &= \mathbf{y}^\top \mathbf{X} - \theta^\top \mathbf{X}^\top \mathbf{X} \\
\theta^\top \mathbf{X}^\top \mathbf{X} &= \mathbf{y}^\top \mathbf{X} \quad | \text{ transpose} \\
\mathbf{X}^\top \mathbf{X} \theta &= \mathbf{X}^\top \mathbf{y} \\
\theta &= \underbrace{(\mathbf{X}^\top \mathbf{X})^{-1}}_{\tilde{p} \times \tilde{p}} \underbrace{\mathbf{X}^\top}_{\tilde{p} \times n} \underbrace{\mathbf{y}}_{n \times 1} \\
& \qquad \underbrace{\qquad\qquad}_{\tilde{p} \times 1}
\end{aligned}
$$

L2-Loss / Squared Error

$L(y, f(\underline{x})) = \frac{1}{2}(y - f(\underline{x}))^2$ or $L(y, f(\underline{x})) = (y - f(\underline{x}))^2$

Properties: Convex; differentiable everywhere

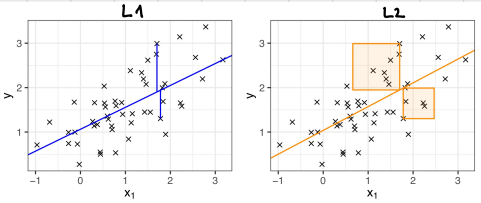
Problem: Outliers have a huge effect on L2.

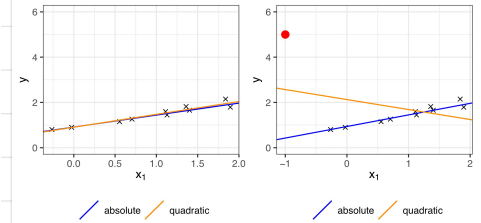
L1-Loss / Absolute Error

$L(y, f(\underline{x})) = |y - f(\underline{x})|$

Properties: Convex; Robust against outliers

Problem: Not differentiable, slower to optimize





Polynomial Regression

General Linear Model

We predict $y \in \mathbb{R}$ as linear combinations of basis functions ϕ_j

$\hat{y} = f(\underline{x}) = \theta_0 + \theta_1 \phi_1(x_1) + \dots + \theta_p \phi_p(x_p)$

If we set $\phi_j \equiv id_x$ we get the usual linear regression model.

! This model is linear in parameters θ but not in coveriates/features $\underline{x}$.

$f(a\underline{x} + b\underline{x}^*; \underline{\theta}) \neq a f(\underline{x}; \underline{\theta}) + b f(\underline{x}^*; \underline{\theta})$ but $f(\underline{x}; a\underline{\theta} + b\underline{\theta}^*) = a f(\underline{x}; \underline{\theta}) + b f(\underline{x}; \underline{\theta}^*)$

The design matrix now is defined as $X = \begin{pmatrix} 1 & | & & | \\ \vdots & \phi_1(\underline{x}_1) & \dots & \phi_p(\underline{x}_p) \\ 1 & | & & | \end{pmatrix}$

Polynomial Regression

In the Polynomial Regression we choose the basis functions $\phi^{(d)}: \mathbb{R} \to \mathbb{R}, \; x_j \mapsto \sum_{k=1}^{d} \beta_k x_j^k$

This results in $f(\underline{x}) = \theta_0 + \theta_1 \phi^{(d)}(\underline{x}) = \theta_0 + \sum_{k=1}^{d} \theta_{1,k} x^k$ where $\theta_{1,k} := \theta_1 \cdot \beta_k$

$X = \begin{pmatrix} 1 & x^{(1)} & (x^{(1)})^2 & \dots & (x^{(1)})^d \\ \vdots & \vdots & & & \vdots \\ 1 & x^{(n)} & (x^{(n)})^2 & \dots & (x^{(n)})^d \end{pmatrix}$, $\underline{\theta} \in \mathbb{R}^{d+1}$

Supervised Classification

Setup

Classification tasks aim at predicting a discrete output $y \in \mathcal{Y} = \{C_1, .., C_g\}$ with $2 \leq g < \infty$ given Data $\mathcal{D}$.

We encode $\mathcal{Y}$ as: • binary case $g=2$: $\mathcal{Y} = \{0,1\}$ or $\mathcal{Y} = \{-1,1\}$.

• multiclass case $g>2$: $\mathcal{Y} = \{1, ..., g\}$

! Our model $f: \mathcal{X} \to \mathbb{R}^g$ outputs scores/probabilities and not classes.

↳ Continuous function is easier to optimize than discrete-valued functions.

↳ Scores/Probabilities contain more information and can be transformed into classes. But this transformation is non-injective

Scoring classifier

Construct g discriminant/scoring functions $f_1, ..., f_g : \mathcal{X} \to \mathbb{R}$.

We choose the class with the maximum score, i.e. $h(\underline{x}) := \underset{k \in \{1,..,g\}}{\arg\max}\ f_k(\underline{x})$

Special case $g=2$:

One discriminant/scoring function is sufficient. $f(\underline{x}) = f_1(\underline{x}) - f_{-1}(\underline{x})$ (Note: $\mathcal{Y} = \{-1,1\}$)

$h(\underline{x}) = \text{sgn}(f(\underline{x}))$ and $|f(\underline{x})|$ is called confidence.

Linear Classifier

If the discriminant/scoring functions $f_1, ..., f_g$ can be specified as linear functions, i.e. $g(f_k(\underline{x})) = \underline{w}_k^T \underline{x} + b_k$ with g being a rank-preserving, monotone transformation, then we call $f_1, ..., f_g$ linear classifiers.

The decision boundaries are hyperplanes.

$\geq$ Probabilistic classifier

Construct g probability functions $\pi_1, ..., \pi_g : \mathcal{X} \to [0,1]$ with $\sum_{k=1}^{g} \pi_k = 1$.

We choose the class with the maximum score, i.e. $h(\underline{x}) := \underset{k \in \{1,..,g\}}{\arg\max}\ \pi_k(\underline{x})$

Special case $g=2$:

One discriminant/scoring function is sufficient. $\pi(\underline{x}) = 1 - \pi_0(\underline{x})$ (Note: $\mathcal{Y} = \{0,1\}$)

$h(x) = \mathbb{1}[\pi(\underline{x}) \geq c]$ with c being a specified threshold.

Decision Region

A decision region for class k is defined as $\mathcal{X}_k := \{\underline{x} \in \mathcal{X} \mid h(\underline{x}) = k\}$

The decision boundaries are defined as hypersurfaces in $\mathcal{X}$ with tied maximal score:

$\{\underline{x} \in \mathcal{X} \mid \exists i \neq j \in \{1,..,g\} : f_i(\underline{x}) = f_j(\underline{x}) \wedge \forall k \neq i,j : f_i(\underline{x}) \geq f_k(\underline{x}) \wedge f_j(\underline{x}) \geq f_k(\underline{x})\}$

Special case $g=2$ with threshold c the decision boundary is $\{\underline{x} \in \mathcal{X} \mid f(\underline{x}) = c\}$

Approaches to construct classifiers

Generative Approach

Idea: Assume that the data generating process for $\underline{x}$ depends on y.

Then we want to answer "Which y tends to have $\underline{x}$ like that?"

$$\pi_k(\underline{x}) = \mathbb{P}(y=k \mid \underline{x}) = \frac{\mathbb{P}(\underline{x} \mid y=k) \cdot \mathbb{P}(y=k)}{\mathbb{P}(\underline{x})} = \frac{\mathbb{P}(\underline{x} \mid y=k) \cdot \pi_k}{\sum_{j=1}^{g} \mathbb{P}(\underline{x} \mid y=j) \cdot \pi_j} \propto \mathbb{P}(\underline{x} \mid y=k) \cdot \pi_k$$

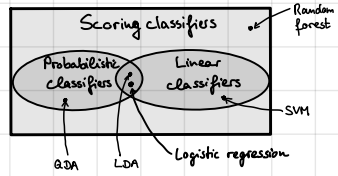
with π_j being a prior estimated from data, by $\hat{\pi}_j = \frac{n_j}{n}$ using e.g. relative frequencies.

Discriminant Approach

Idea: Use empirical risk minimization with a suitable Loss-function

We want to answer "What is the best prediction for y given $\underline{x}$?".

$$\hat{f} = \underset{f \in \mathcal{H}}{\arg\min}\ R_{emp}(f) = \underset{f \in \mathcal{H}}{\arg\min} \sum_{i=1}^{n} L(y^{(i)}, f(\underline{x}^{(i)}))$$



Examples

Logistic Regression (Discriminant)

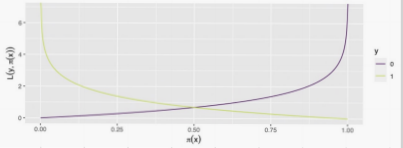
$\mathcal{H} = \{\pi : \mathcal{X} \to [0,1] \mid \pi(\underline{x}) = s(\underline{\theta}^T \underline{x})\}$

with $s(\underline{\theta}^T \underline{x}) = \frac{\exp(\underline{\theta}^T \underline{x})}{1 + \exp(\underline{\theta}^T \underline{x})} = \frac{1}{1 + \exp(-\underline{\theta}^T \underline{x})}$

Logistic / Bernoulli Loss function:

$L(y, \pi(\underline{x})) = -y \log(\pi(\underline{x})) + (1-y) \log(1 - \pi(\underline{x}))$

Optimization is done numerically (GD).



Penalizes heavier the more confident we are in a wrong prediction.

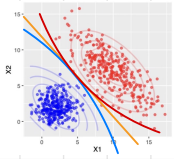
Discriminant Analysis (Generative)

Linear discriminant analysis assumes that each class density is modelled as $\underline{x} | y=k \sim N(\underline{\mu}_k, \Sigma)$

where the covariance Σ is equal for all classes.

Note: $\pi_k(\underline{x}) \propto p(\underline{x} | y=k) \cdot \pi_k \propto \exp(\theta_{0k} + \underline{x}^T \underline{\theta}_k)$

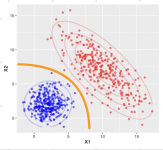
$\Rightarrow \log(\pi_k(\underline{x})) = \theta_{0k} + \underline{x}^T \underline{\theta}_k \Rightarrow$ LDA gives a linear classifier



Quadratic discriminant analysis assumes that each class density is modelled as $\underline{x} | y=k \sim N(\underline{\mu}_k, \Sigma_k)$

where the covariances Σ_k do not need to be equal.

Note: $\log(\pi_k(\underline{x}))$ defines a quadratic function in $\underline{x}$, i.e. not linear.

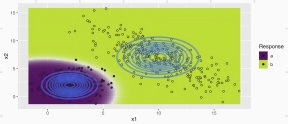


Naive Bayes (Generative)

In Naive Bayes we assume that given class y the features are conditionally independent, i.e. $p(\underline{x} | y=k) = \prod_{j=1}^{p} p(x_j | y=k)$

• For numerical features: We further assume that $x_j | y=k \sim N(\mu_{jk}, \sigma_{jk}^2)$. Gaussian Naive Bayes (GNB)

This results in $\underline{x} | y=k \sim N(\underline{\mu}_k, D_k)$ with D_k being a diagonal covariance matrix. $\Rightarrow$ GNB is a special case of QDA.



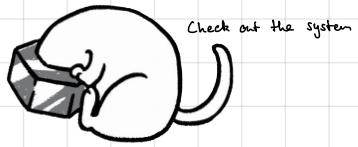
• For categorical features: We further assume that $x_j | y=k$ is multinomial distributed, i.e. $p(x_j | y=k) \propto \prod_{m=1}^{g} p_{kjm}^{n_m}$

with n_m / p_{kjm} being the absolute / relative frequency of value m in feature j restricted to class k. Multinomial Naive Bayes

Note: The multinomial Naive Bayes classifier is a linear classifier.

! If $\exists m : p_{kjm} = 0$, then we replace the 0 with a small value (Laplace Smoothing).

Performance Evaluation



Generalization Error

Generalization Error for a Fixed Model

Let $(\underline{x}, y) \sim \mathbb{P}_{xy}$, $\hat{f}$ a fixed model and L a Loss-function.

We define the Generalization error of a fixed model $\hat{f}$ as

$$GE(\hat{f}, L) := \mathbb{E}_{xy}\left[L(y, \hat{f}(\underline{x}))\right]$$

If there exists a dedicated test set $\mathcal{D}_{test} \in \mathbb{D}_n$ then we can estimate $\widehat{GE}(\hat{f}, L) := \frac{1}{m} \sum_{(\underline{x},y) \in \mathcal{D}_{test}} L(y, \hat{f}(\underline{x}))$

! Rarely possible, because at the end we fit our model on the complete data set. So no data is left available for $\mathcal{D}_{test}$.

Example

$MSE(\hat{f}) = \mathbb{E}\left[(y - \hat{f}(\underline{x}))^2\right]$ is a GE of $\hat{f}$ with L2-Loss.

$$\mathbb{E}\left[(y - \hat{f}(\underline{x}))^2\right] = Var(\hat{f}(\underline{x})) + Bias(\hat{f}(\underline{x}))^2 + \underbrace{Var(\varepsilon)}_{\text{irreducible noise}}$$

Setup

We are interested in evaluating our (final) model $\hat{f}$. But because this is rarely possible we evaluate the next best thing, i.e. the inducer.

To do so, we might need to define specific measures.

Inner and Outer Loss

Sometimes we want to use a different performance measure to evaluate the model/inducer, then we used to construct the model, i.e. the Loss-function.

We call the Loss-function that we used for ERM inner loss.

We call the performance evaluation function outer loss.

! The outer loss doesn't need to be a pointwise Loss-function

Set-based Performance Metrics

Sometimes we evaluate a model performance based on the set of predictions as a whole and not on observation based losses. We use a metric

$\rho : \bigcup_{m \in \mathbb{N}} (\mathbb{Y}^m \times \mathbb{R}^{m \times g}) \to \mathbb{R}$, $(y, F) \mapsto \rho(y, F)$, where F is defined as $F_{\mathcal{D}_{test}, \hat{f}} := \begin{bmatrix} \hat{f}(\underline{x}^{(1)}) \\ \vdots \\ \hat{f}(\underline{x}^{(m)}) \end{bmatrix} \in \mathbb{R}^{m \times g}$ and $|\mathcal{D}_{test}| = m$.

Pointwise loss L can be seen as a special case of a metric ρ with $\rho_L(y, F) = \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, F^{(i)}) = \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, \underbrace{\hat{f}(\underline{x}^{(i)})}_{=\hat{y}^{(i)}})$

Example

In Logistic Regression we use the Logistic/Bernoulli Loss-function as inner Loss.

For performance evaluation, i.e. outer loss, we can use the AUC.

Connection between both GE's

We can only estimate GE with $n_{train} = |\mathcal{D}_{train}| < |\mathcal{D}| = n$ which could result in a pessimistic bias for $\widehat{GE}$, i.e. we overestimate the GE.

Using a pointwise losses ρ_L we can rewrite the GE for an inducer as

$$GE(\mathcal{I}, \lambda, n_{train}, \rho_L) = \mathbb{E}_{\mathcal{D}_{train}, (\underline{x}, y)}\left[L(y, \underbrace{\mathcal{I}(\mathcal{D}_{train}, \lambda)}_{=\hat{f} \in \mathcal{H}}(\underline{x})\right]$$

! No limit, b.c. that was only needed for set-based-performance-metric.

Using that, we can rewrite the GE for a fixed model $\hat{f}$ that was trained on $\mathcal{D}_{train} = \mathcal{D}$, i.e. $\hat{f} = \mathcal{I}(\mathcal{D}_{train}, \lambda)$, as

$GE(\hat{f}, L) = GE(\mathcal{I}, \lambda, n_{train}, \rho_L \mid \mathcal{D}_{train})$ or equivalently

$$\mathbb{E}_{\underline{x},y}\left[L(y, \hat{f}(\underline{x}))\right] = \mathbb{E}_{\mathcal{D}_{train}, (\underline{x},y)}\left[L(y, \mathcal{I}(\mathcal{D}_{train}, \lambda)(\underline{x}) \mid \mathcal{D}_{train}\right]$$

Generalization Error for Inducer

Let $\mathcal{I}_\lambda$ be an Inducer on n_{train} points from $\mathbb{P}_{xy}$ and let ρ be a performance metric (outer loss).

We define the Generalization Error for the Inducer $\mathcal{I}$ as

$$GE(\mathcal{I}, \lambda, n_{train}, \rho) := \lim_{n_{test} \to \infty} \mathbb{E}_{\mathcal{D}_{train}, \mathcal{D}_{test}}\left[\rho(\underline{y}, F_{\mathcal{D}_{test}, \mathcal{I}(\mathcal{D}_{train}, \lambda)})\right]$$

with $n_{train} = |\mathcal{D}_{train}|$.

Idea: Quality of models when fitted with $\mathcal{I}_\lambda$ on n_{train} points from $\mathbb{P}_{xy}$.

! Expected value over $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$ sampled independently. Rarely possible, therefore we use resampling methods to obtain an estimate for $GE(\mathcal{I}, \lambda, n_{train}, \rho)$.

! Some ρ might only converge for $n_{test} \to \infty$, which is why we need the limit and why we take the expectation over $\mathcal{D}_{test} \in \mathbb{D}_{n_{test}}$ instead of $(\underline{x}, y) \sim \mathbb{P}_{xy}$

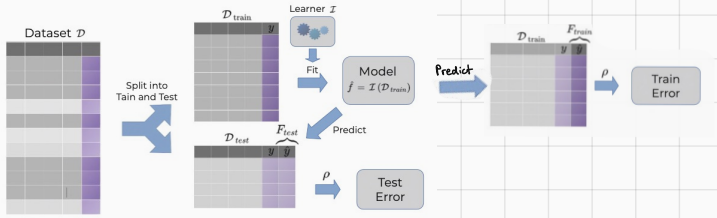
Training- and Test-Error

Training Error and Test Error

Let $\hat{f} = \mathcal{I}(\mathcal{D}_{train}, \lambda)$ and let ρ be a performance metric. We define the

- Training Error as $\rho\left(\underline{y}_{train}, F_{\mathcal{D}_{train}, \hat{f}}\right)$
- Test Error as $\rho\left(\underline{y}_{test}, F_{\mathcal{D}_{test}, \hat{f}}\right)$

! Choose $\mathcal{D}_{test}$ and $\mathcal{D}_{train}$, s.t. $\mathcal{D}_{test} \cap \mathcal{D}_{train} = \emptyset$ to avoid optimistic Bias.



Train Error vs. Test Error

Goodness-of-fit measures - like R^2, $\mathcal{L}$, AIC, BIC, deviance - is based on training error but are based on distributional assumptions and limited data - therefore hard to use for high-dimensional or more complex data.

Decrease of n_{train} ⇒ Increase of Test Error (in general) b.c. model generalizes better with more training data and worse with less training data.

Increase of complexity ⇒ Decrease of Training Error (in general) b.c. it becomes easier to learn all patterns on small training data sets or with more flexibility.

Decrease of n_{test} ⇒ Increase of variance of test error.

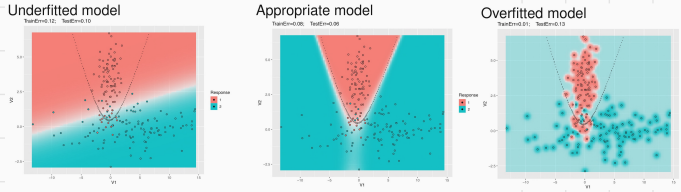
Overfitting and Underfitting

Underfitting occurs when a model can't reflect the true shape of underlying function (given the data)

↳ High Test Error and high Training Error.

Overfitting occurs when a model reflects noise or artifacts from $\mathcal{D}_{train}$ which do not generalize.

↳ Small Train Error and high Test Error.



Overfitting is influenced by

- Complexity of Model, i.e $\dim(\Theta)$
- Amount of Training Data, i.e. n_{train}
- Dimensionality of Feature Space, i.e. $\dim(\mathcal{X})$
- Irreducible Noise, i.e. aleatoric uncertainty

Overfitting can be avoided by regularization methods:

- Constrain $\mathcal{H}$ by using less complex model classes
- Higher amount or better quality of training data
- Use Feature Selection/Engineering and keep only the features that carry a lot of useful information.
- Occam's razor: If two models have similar GE, prefer the simpler one.

Bias-Variance-Tradeoff

The GE of the inducer can be estimated by the Test Error.

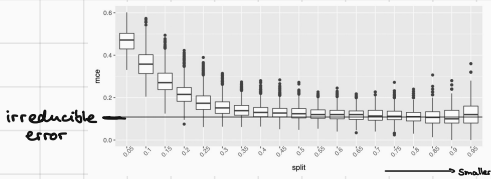
The quality of this estimate is based on the Bias(*) and the Variance of the Test Error. (*) Smaller Test Error means less Bias.

Those are influenced by the complexity and amount of training data.

Influence of n_{train}:

Because in practice $n = |\mathcal{D}|$ is fixed increase/decrease of n_{train} results in a decrease/increase of n_{test}. So

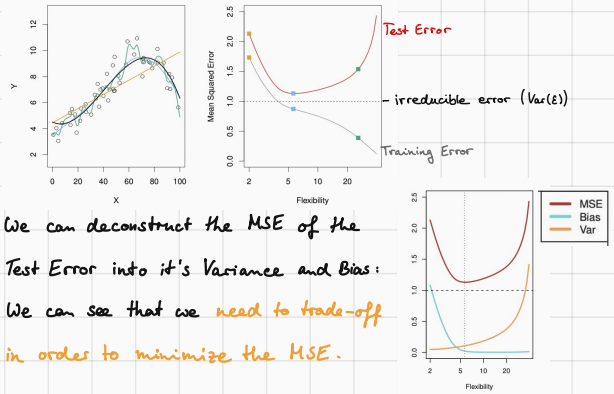
Increasing n_{train} leads to decrease in Test Error but higher variance of Test Error.



Rule of thumb for hold-out split:

$\frac{2}{3}$ of $\mathcal{D}$ as $\mathcal{D}_{train}$ and $\frac{1}{3}$ of $\mathcal{D}$ as $\mathcal{D}_{test}$

Influence of complexity:



We can deconstruct the MSE of the Test Error into it's Variance and Bias:

We can see that we need to trade-off in order to minimize the MSE.

Resampling

← Looks like a Dumpling which rhymes with Resampling

Setup

We want to estimate $GE(I, \lambda, n, \rho_L) = GE(\hat{f}, L)$

but doing so with a single hold-out-split results in a high pessimistic Bias or high Variance.

So instead we split repeatedly and average the result.

This way we reduce the variance from small test sets.

Resampling Strategies

We choose $J_{train} \in \{1,\dots,n\}^{n_{train}}$ and $J_{test} \in \{1,\dots,n\}^{n_{test}}$.

"Choose n_{train} indices from $\{1,\dots,n\}$ with replacing"

We define $\mathcal{J} := \left((J_{train,1}, J_{test,1}), \dots, (J_{train,B}, J_{test,B}) \right)$

with $B \in \mathbb{N}$ and $n_{train,1} \approx n_{train,2} \approx \dots \approx n_{train,B} \approx n_{train}$

Our estimate is then

$$\widehat{GE}(I, \lambda, \mathcal{J}, \rho) = \frac{1}{B} \sum_{k=1}^{B} \rho\left(\underline{y}_{J_{test,k}}, F_{J_{test,k}, I(D_{train,k}, \lambda)} \right)$$

Bias-Variance Analysis in Resampling

If there exists a dedicated Test set that is not used to train $\hat{f}$, then $\widehat{GE}(\hat{f}, L) = \frac{1}{m} \sum_{(x,y) \in D_{test}} L(y, \hat{f}(x))$

with the m samples being i.i.d. and $\mathbb{E}[\widehat{GE}(\hat{f}, L)] = \mathbb{E}[L(y, \hat{f}(x))] = GE(\hat{f}, L)$ and $\mathbb{V}[\widehat{GE}(\hat{f}, L)] = \frac{1}{m} \mathbb{V}[L(y, \hat{f}(x))]$

We can apply the Central Limit Theorem to approx. the distr. of $\widehat{GE}(\hat{f}, L)$ and compute Confidence intervals.

If $\hat{f}$ is trained on D we estimate $GE(I, \lambda, n, \rho)$ instead of $GE(\hat{f}, L)$ using a resampling based estimate $\widehat{GE}(I, \lambda, \mathcal{J}, \rho)$.

$$\mathbb{E}[\widehat{GE}(I, \lambda, \mathcal{J}, \rho_L)] \approx \mathbb{E}\left[\rho_L\left(\underline{y}_{J_{test}}, F_{J_{test}, I(D_{train}, \lambda)} \right)\right] = \mathbb{E}\left[\frac{1}{m} \sum_{(x,y) \in D_{test}} L(y, I(D_{train})(x))\right] = GE(I, \lambda, n_{train}, \rho_L)$$

! So our estimate for $GE(I, \lambda, n, \rho)$ is in expectation nearly correct, i.e. n_{train} instead of n. B.c. $n_{train} < n$ our estimate is pessimistically biased - on n it would perform better.

($\rightarrow$ i.e. unbiased)

Cross Validation

1. Split the data into k equally-sized partitions.
   - 1.1. Stratification. For unbalanced target classes perform the split per class and rejoin the data. Preserves the target distr.
2. Each set is test set once, the rest is used for training $\rightarrow$ Results in k test errors (and training errors)
3. Average the test errors

! 5 or 10 fold CV's are common. n-fold-CV is also called Leave-one-out-CV (LOO-CV) $\rightarrow$ Nearly unbiased but high variance

! The test errors are not independent b.c. the splits aren't independent $\rightarrow$ There is no unbiased estimate for $\mathbb{V}[\widehat{GE}(I, \dots)]$.

Subsampling / Monte Carlo CV

1. Split the data into two partitions
2. Use one set for training and the other for testing. $\rightarrow$ Results in one test error (hold-out split)
3. Repeat step 1. & 2. many times
4. Average over all test errors

! $\frac{8}{10}$ or $\frac{9}{10}$ of the data for training is common.

Bootstrap

1. Draw a trainingsets of size n with replacement from the full data set D. The data of D that is not drawn is the test set.
2. Train and evaluate the model $\rightarrow$ Results in a test error
3. Repeat 1. & 2. many times
4. Average over all test errors

! A training set contains about $\frac{2}{3}$ unique points (for $n \to \infty$)

Leave-one-Object-Out

This method is used when there are multiple observations per object $\rightarrow$ Data is not i.i.d. anymore.

The data of one object should either be in the training set or test set ! Not in both $\rightarrow$ Otherwise $\widehat{GE}$ is biased.

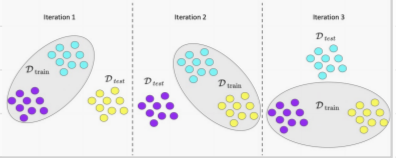
This can be achieved by using CV on objects, i.e. split the objects into k groups...

Example CV

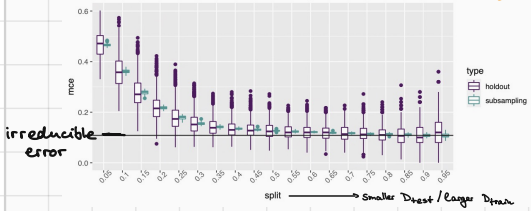
3-fold CV

stratified 3-fold cross-validation

Leave-One-Object-Out



Bias-Variance-Trade off in Subsampling

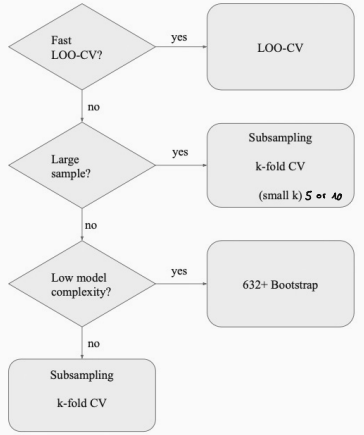


! $\text{Bias}(\text{Subsampling} \mid \text{split-rate}) \overset{!}{=} \text{Bias}(\text{Hold-out} \mid \text{split-rate})$

$\text{Variance}(\text{Subsampling} \mid \text{split-rate}) < \text{Variance}(\text{Hold-out} \mid \text{split-rate})$

$\Rightarrow$ "optimal" split-rate is higher in Subsampling compared to one Hold-out-split.

Guidelines



Measures for Regression

Pointwise outer-Loss

Mean squared error (MSE):

$$\rho_{MSE}(\vec{y}, \vec{F}) = \frac{1}{m} \cdot \sum_{i=1}^{m} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 \in [0, +\infty) \rightarrow \text{L2 loss}$$

Similar: Sum of squared errors (SSE) $\rho_{SSE} = m \cdot \rho_{MSE}$

root MSE (RMSE) $\rho_{RMSE} = \sqrt{\rho_{MSE}}$

Mean absolute error (MAE):

$$\rho_{MAE}(\vec{y}, \vec{F}) = \frac{1}{m} \cdot \sum_{i=1}^{m} \left|y^{(i)} - \hat{y}^{(i)}\right| \in [0, +\infty) \rightarrow \text{L1 loss}$$

Similar: median absolute error ρ_{MedAE} (even more robust)

Mean absolute percentage error (MAPE):

$$\rho_{MAPE}(\vec{y}, \vec{F}) = \frac{1}{m} \cdot \sum_{i=1}^{m} \left|\frac{y^{(i)} - \hat{y}^{(i)}}{y^{(i)}}\right| \in [0, +\infty)$$

↳ Small $|y|$ influence more, but can't handle $y = 0$

Set-based outer-Loss

R^2:

$$\rho_{R^2}(\vec{y}, \vec{F}) = 1 - \frac{\sum_{i=1}^{m} \left(y^{(i)} - \hat{y}^{(i)}\right)^2}{\sum_{i=1}^{m} \left(y^{(i)} - \bar{y}\right)^2} = 1 - \frac{SSE_{LinModel}}{SSE_{ConstantModel}}$$

$$= 1 - \frac{\rho_{MSE}(\vec{y}, LM(\vec{y}))}{\rho_{MSE}(\vec{y}, \bar{y})}$$

! Higher $R^2 \not\Rightarrow$ Better fit.

But R^2 is invariant w.r.t linear scaling of y. MSE is not.

Generalized R^2:

$$1 - \frac{Loss_{ComplexModel}}{Loss_{SimpleModel}}$$

E.g. model vs. constant, linear vs. non-linear, tree vs. forest, ...

! Usually $R^2 \in [0,1]$, but this is only true if we evaluate on Training Data. On Test Data $R^2 < 0$ is possible → Overfitting

Spearman's ρ:

$$\rho_{Spearman}(\vec{y}, \vec{F}) = \frac{Cov(rg(\vec{y}), rg(\vec{y}))}{\sqrt{Var(rg(\vec{y})) \cdot Var(rg(\vec{y}))}} \in [-1, 1]$$

↳ Very robust against outliers and invariant under monotone transformations of $\vec{y}$.

Measures for Classification

Confusion Matrix

For multiclass classification

True classes

Predicted classes	setosa	versicolor	virginica	error	n
setosa	50	0	0	0	50
versicolor	0	46	4	4	50
virginica	0	4	46	4	50
error	0	4	4	8	-
n	50	50	50	-	150

For binary classification:

Pred. \ True Class	y positiv	y negativ	
$\hat{y}$ pos.	True positiv	False positive	#Pred. pos.
$\hat{y}$ neg.	False negative	True negative	#Pred. neg.
	#Class positiv	#Class negativ	n

Set-based outer-Loss / ROC Analysis

True condition	Condition positive	Condition negative		
Total population			Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$
Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{TPR}{FPR}$	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$	F_1 score = $\frac{1}{\frac{1}{Recall} + \frac{1}{Precision}} = \frac{2TP}{2TP + FN + FP}$
False negative rate (FNR), Miss rate = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{FNR}{TNR}$		

! Not a good performance measure when label distr. is unbalanced.
Accuray paradox

→ Usually preferable in situations with imbalanced data.

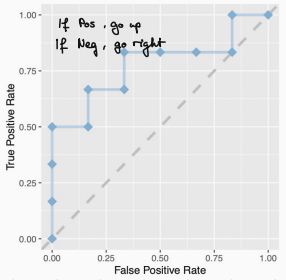
→ Balances TPR and PPV but tends to the smaller value.

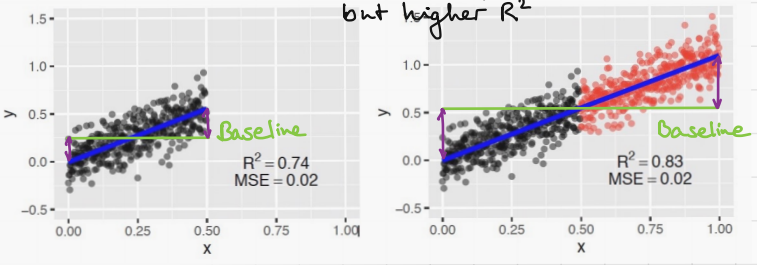
Different metrics emphasize different aspects of performance. Choice requires Domain Knowledge.

DRAWING ROC CURVES

Sorted

#	Truth	Score
1	Pos	0.95
2	Pos	0.86
3	Pos	0.69
4	Neg	0.65
5	Pos	0.59
6	Neg	0.52
7	Pos	0.51
8	Neg	0.39
9	Neg	0.28
10	Neg	0.18
11	Pos	0.15
12	Neg	0.06





Pointwise outer-Loss using class-labels

Misclassification error rate (MCE):

$$\rho_{MCE} = \frac{1}{m} \cdot \sum_{i=1}^{m} \left[y^{(i)} \neq \hat{y}^{(i)}\right] = \frac{\sum \text{False pos.} + \sum \text{False neg.}}{\text{Total population}}$$

Accuracy (ACC):

$$\rho_{ACC} = \frac{1}{m} \cdot \sum_{i=1}^{m} \left[y^{(i)} = \hat{y}^{(i)}\right] = \frac{\sum \text{True pos.} + \sum \text{True neg.}}{\text{Total population}}$$

! No information about how good/skewed prob's are. Errors on all classes weighted equally, which is often inappropriate

Cost matrix:

$$\text{Costs} = \frac{1}{n} \cdot \sum_{i=1}^{n} C\left[y^{(i)}, \hat{y}^{(i)}\right] = \frac{1}{n} \langle \text{Cost Matrix}, \text{Confusion Matrix} \rangle_F$$

Example:

Cost matrix

		True classes: solvent	True classes: not solvent
Predicted classes	solvent	0	10,000
	not solvent	100	0

Confusion matrix

		True classes: solvent	True classes: not solvent
Predicted classes	solvent	70	3
	not solvent	7	20

$$\text{Costs} = \frac{1}{n} \langle \text{Cost Matrix}, \text{Confusion Matrix} \rangle_F = \frac{1}{100} \cdot tr\left(\begin{pmatrix} 0 & 10000 \\ 100 & 0 \end{pmatrix}^T \begin{pmatrix} 70 & 3 \\ 7 & 20 \end{pmatrix}\right) = \frac{1}{100} tr\begin{pmatrix} 700 & - \\ - & 30000 \end{pmatrix} = 307$$

Pointwise outer-Loss using probabilities

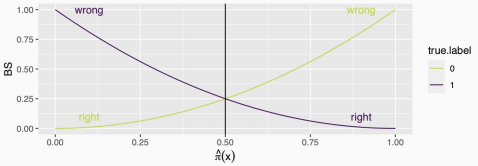
Brier Score:

$$\rho_{BS} = \frac{1}{m} \cdot \sum_{i=1}^{m} \left(y^{(i)} - \hat{\pi}^{(i)}\right)^2 \quad \text{(MSE for probabilities)}$$

↳ $\in \{0,1\}$

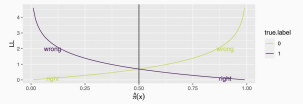
Multi-Class Brier Score: ↳ $:= [y^{(i)} = k] \in (0,1)$

$$\rho_{BS,MC} = \frac{1}{m} \cdot \sum_{i=1}^{m} \sum_{k=1}^{g} \left(o_k^{(i)} - \hat{\pi}_k^{(i)}\right)^2$$



Log-Loss:

$$\rho_{LL} = \frac{1}{m} \sum_{i=1}^{m} \left(-y^{(i)} \log\left(\hat{\pi}^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \hat{\pi}^{(i)}\right)\right).$$



Optimal value is 0, "confidently wrong" is penalized heavily.

ROC - Curves

ROC - Kurve

Die ROC - Kurve zeigt die Zuverlässigkeit der Vorhersagen für alle möglichen Schwellenwerte c an.

Verbindet die Punkte $(FPR(c), TPR(c))$ $\forall c \in [x_{(1)}, x_{(n)}]$

Für $c < x_{(1)} \Rightarrow \hat{y}_i = 1 \ \forall i \Rightarrow (FPR(c), TPR(c)) = (1,1)$. Für $c > x_{(n)} \Rightarrow \hat{y}_i = 0 \ \forall i \Rightarrow (FPR(c), TPR(c)) = (0,0)$

AUC - Maß

Maß zur Bewertung der ROC Kurve. $AUC := \frac{N_C + \frac{N_E}{2}}{N}$

↳ Fläche unter ROC - Kurve.

, N_C := Anzahl konkordanter Paare

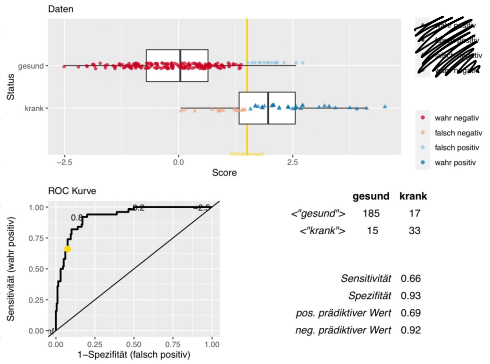
, N_E := Anzahl Paare mit Bindung in X. $= |\{(i,j) : x_i = x_j, y_i \neq y_j\}|$

> perfekte Trennbarkeit der Y-Gruppen durch c bedeutet $AUC = 1$

> X, Y unabhängig $\Rightarrow AUC \approx 0$

! Behandelt Spezifität & Sensitivität gleichwertig und beachtet positiv/negativ prädikaten Wert nicht.

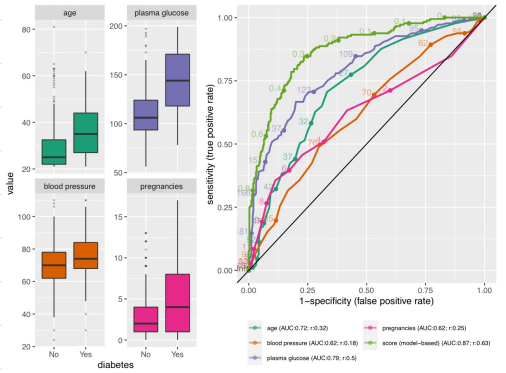
Beispiel



	gesund	krank
<"gesund">	185	17
<"krank">	15	33

Sensitivität	0.66
Spezifität	0.93
pos. prädiktiver Wert	0.69
neg. prädiktiver Wert	0.92

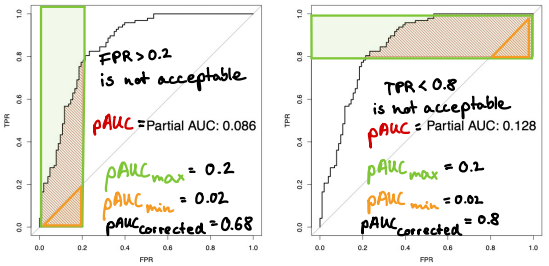
Beispiel



Partial AUC

If a certain value for TPR or FPR is not acceptable, then it might be usefull to fix TPR or FPR to a required value and optimize the other.

We then don't want to account for the cut-of region in our AUC. Hence, partial AUC.



Integrate over the region that is left after cut-off, i.e. the red-region.

! Problem: partial AUC depends on cut-off-value which makes it less interpretable.

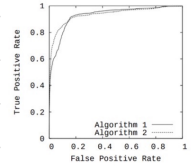
Solution: Corrected partial AUC. We normalize the partial AUC to $[0,1]$.

$$pAUC_{corrected} = \frac{1}{2}\left(1 + \frac{pAUC - pAUC_{min}}{pAUC_{max} - pAUC_{min}}\right)$$

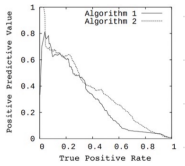
Precision - Recall Curves

Create a ROC-like plot, but with TPR and PPV instead of TPR and FPR.

might be better for highly imbalanced data ($n_- \gg n_+$)



(a) Comparison in ROC space



(b) Comparison in PR space

> Both learners seem to perform well under ROC. But under PR there is visible room for improvement.

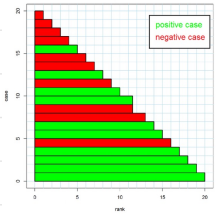
> PR reveals better, that Algo 2 is superior to Algo 1.

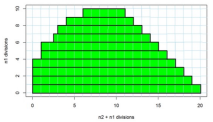
> PR is more sensitive to imbalanced classes.

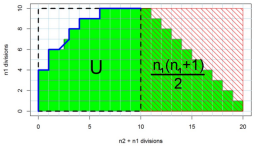
Curve dominates fully in ROC $\Leftrightarrow$ Curve dominates fully in PR.

AUC & Mann - Whitney - U Test

Next, keep only the green ones, and slide them horizontally to get a nice even stairstep on the right edge:





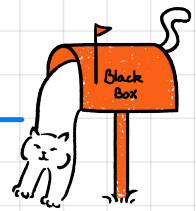


- U: area of the green bars on the left.
- $n_+ \cdot n_-$: area of the dashed rectangle.

$\Rightarrow$ AUC is U normalized to the unit square:

$$AUC = \frac{U}{n_+ \cdot n_-}.$$

Important Learners in Machine Learning



k-Nearest-Neighbours

Setup

k-NN generates predictions for $\underline{x}$ based on the target-values of its k closest neighbours $N_k(\underline{x})$

! k-NN can be used as a classifier and for regression

k-Neighbourhood

Let $(\mathcal{X}, d)$ be a metric space, i.e. d is a distance measure on the feature space $\mathcal{X}$.

$N_k(\underline{x}) := \{ \underline{x}^{(i)} \in \mathcal{X} \mid d(\underline{x}^{(i)}, \underline{x}) \leq d(\underline{x}^{(k)}, \underline{x}) \}$

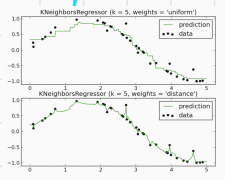
We define $I_k(\underline{x}) := \{ i \in \mathbb{N} \mid \underline{x}^{(i)} \in N_k(\underline{x}) \}$

Prediction Regression

$$\hat{f}(\underline{x}) = \frac{1}{k} \sum_{i \in I_k(\underline{x})} y^{(i)}$$

$$\hat{f}(\underline{x}) = \frac{1}{\sum_{i \in I_k(\underline{x})} \omega^{(i)}} \cdot \sum_{i \in I_k(\underline{x})} \omega^{(i)} y^{(i)} \quad \text{with} \quad \omega^{(i)} = \frac{1}{d(\underline{x}^{(i)}, \underline{x})}$$

Example



Standardization and Weights

- Features in k-NN are usually standardized or normalized.
  because most distances place higher importance on features with higher ranges.
- Some Features can be given a higher Importance by using a weighted distance measure.

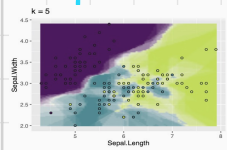
$$d^{\text{weighted}}_{\text{Euclidean}}(\underline{x}, \tilde{\underline{x}}) = \sqrt{\sum_{j=1}^{p} w_j (x_j - \tilde{x}_j)^2}$$

Prediction Classification

$$\hat{\pi}_\ell(\underline{x}) = \frac{1}{k} \sum_{i \in I_k(\underline{x})} \mathbb{1}[y^{(i)} = \ell]$$

$$\hat{h}(\underline{x}) = \underset{\ell \in \{1, \dots, g\}}{\arg\max}\ \hat{\pi}_\ell(\underline{x})$$

Example



Gower Distance

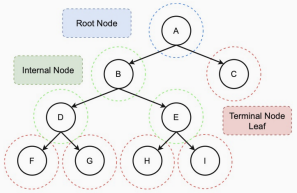
Metric that measures the dissimilarity of two items with mixed numeric and non-numeric data.

$$d_{\text{Gower}}(\underline{x}^{(i)}, \underline{x}^{(j)}) = \frac{\sum_{k=1}^{p} \delta_{ijk} \cdot d_{ijk}}{\sum_{k=1}^{p} \delta_{ijk}} \quad \text{with} \quad \delta_{ijk} := \begin{cases} 0, & \text{if } x_k^{(i)} = x_k^{(j)} = 0 \ \vee\ x_k^{(i)} = \text{NA} \ \vee\ x_k^{(j)} = \text{NA} \\ 1, & \text{otherwise} \end{cases}$$

for non-numeric $\underline{x}_k$ $d_{ijk} = \begin{cases} 0, & \text{if } x_k^{(i)} = x_k^{(j)} \\ 1, & \text{otherwise} \end{cases}$ for numeric $\underline{x}_k$ $d_{ijk} = \frac{|x_k^{(i)} - x_k^{(j)}|}{R_k}$, R_k = Range

Classification and Regression Trees

Binary Tree



E is called a parent node of the child nodes H and I

Setup

We use binary Trees to divide the feature space $\mathcal{X}$ into rectangular regions

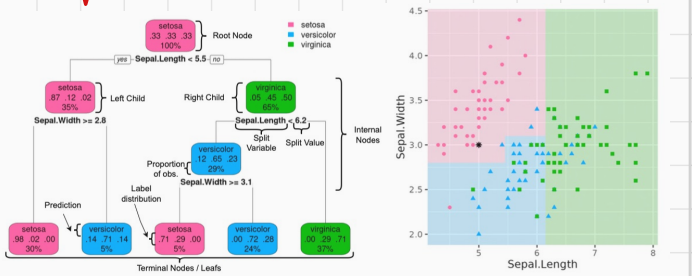
$$f(\underline{x}) = \sum_{m=1}^{M} c_m \cdot \mathbb{1}_{Q_m}(\underline{x}) \quad \text{with}$$

M = # Leaf nodes

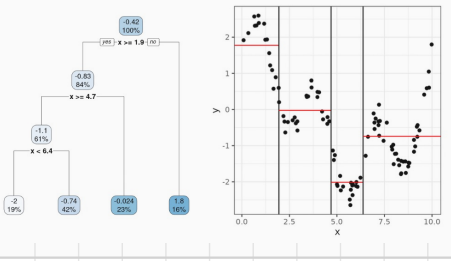
Q_m defines a hyperrectangle in $\mathcal{X}$

c_m is the predicted numerical value or class label

Classification Trees



Regression Trees



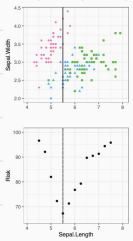
Dealing with Categorical Features

If x_j is categorical with m levels, then there are $2^{m-1} - 1$ possible partitions.

That's very fast too large. For some special cases there exist shortcuts:

- binary classification:
  1. Calculate the proportion of 1-outcomes for each category of the feature in $\mathcal{N}$.
  2. Sort the categories according to these proportions.
  3. The feature can then be treated as if it was ordinal, so we only have to investigate at most $m - 1$ splits.
- regression with L2-Loss:
  1. Calculate the mean of the outcome in each category
  2. Sort the categories by increasing mean of the outcome
  3. The feature can then be treated as if it was ordinal, so we only have to investigate at most $m - 1$ splits.

Best Split for Classification

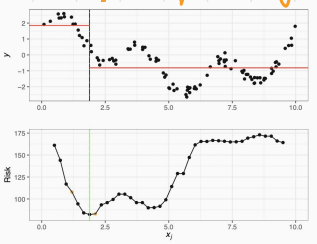


$c_{\mathcal{N}}$ is the predicted constant in node $\mathcal{N}$, i.e. $c_{\mathcal{N}} := (\hat{\pi}_1^{(\mathcal{N})}, \dots, \hat{\pi}_g^{(\mathcal{N})})$

with $\hat{\pi}_k^{(\mathcal{N})}$ being the proportion of class k in $\mathcal{N}$, i.e. $\hat{\pi}_k^{(\mathcal{N})} := \frac{1}{|\mathcal{N}|} \sum_{(y,\underline{x}) \in \mathcal{N}} \mathbb{1}(y = k)$

The Risk is (typically) defined based on either the Multi-class Brier-score, i.e. $L(y, \hat{\underline{\pi}}) = \sum_{k=1}^{g} (\pi_k - o_k(y))^2$ or the Multi-class Log-Loss $L(y, \underline{\pi}) = -\sum_{k=1}^{g} o_k(y) \log(\hat{\pi}_k)$

Best Split for Regression



$c_{\mathcal{N}}$ is the predicted constant in node $\mathcal{N}$,

i.e. $c_{\mathcal{N}} := \underset{c \in \mathbb{R}}{\arg\min} \sum_{(y,\underline{x}) \in \mathcal{N}} L(y, c)$

$\mathcal{R}(\mathcal{N}) := \sum_{(\underline{x},y) \in \mathcal{N}} L(y, c_{\mathcal{N}})$

$\mathcal{N}_1 = \{(\underline{x}, y) \in \mathcal{N} \mid x_j \leq t\}$ and $\mathcal{N}_2 = \mathcal{N} \setminus \mathcal{N}_1$

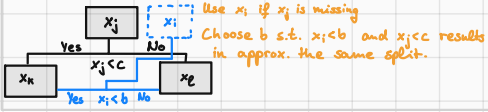
$\mathcal{R}(\mathcal{N}, j, t) := \mathcal{R}(\mathcal{N}_1) + \mathcal{R}(\mathcal{N}_2)$

We want to find $\underset{j,t}{\arg\min}\ \mathcal{R}(\mathcal{N}, j, t)$

Surrogate Split

If we have missing data in the feature x_j, then our Tree might not be able to use that observation.

Solution: Try to mimic the outcome of the split $x_j < c$ by using another feature x_i (or multiple)



Stopping Criteria

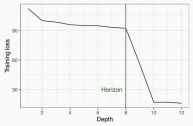
We can define different **stopping criteria**, e.g.: Don't split a node if

- a certain number of leaves if reached,
- it contains too few observations,
- splitting results in children with too few observations,
- splitting does not achieve a certain minimal improvement of the risk in the children, compared to the risk in the parent node,
- it already has the same target value (**pure node**) or identical feature values for all observations.

Selection of a stopping criterion and its concrete values are hyperparameters of CART.

Horizon Effect

Before we try all possible additional splits further down a branch, we can't know whether any one of them will be able to reduce the risk by a lot.

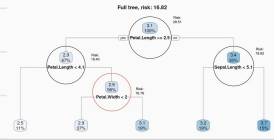


Cost-Complexity Pruning

Idea: Grow a large Tree, then remove the branches with the least informative leaves (repeatedly)

$$R_{reg}(T) := \sum_{m=1}^{|T|} \sum_{i: x^{(i)} \in Q_m} L(y^{(i)}, c_m) + \alpha |T|$$

$|T|$: number of leaves



Random Forests

Bagging

Bootstrap Aggregation is an ensemble method, i.e. it combines many models into one meta-model

↓ called base learners

In Bagging all base learners are of the same type and only differ in their training data, i.e. we train base learners $b^{[m]}(x)$, $m = 1, \dots, M$ on M bootstrap samples of training data D.

We then aggregate (averaging (Regr.) or majority vote (Classif.)) the predictions of $b^{[1]}, \dots, b^{[M]}$ to get the ensemble model $\hat{f}^{[M]}(x)$.

When does Bagging help

We are interested in the instability of the ensemble which we define as $\Delta(\hat{f}^{[M]}(x)) = \frac{1}{M} \sum_{m=1}^{M} (b^{[m]}(x) - \hat{f}^{[M]}(x))^2$

Using this we can derive:

$$\mathbb{E}_{x,y}\left[L(y, \hat{f}^{[M]}(x))\right] = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{x,y}\left[L(y, b^{[m]}(x))\right] - \mathbb{E}_{x,y}\left[\Delta(\hat{f}^{[M]}(x))\right]$$

Under some assumptions we can further derive:

$$\mathbb{E}_{x,y}\left[\Delta(\hat{f}^{[M]}(x))\right] \approx \frac{M-1}{M} \mathbb{V}_{x,y}\left[b^{[m]}(x)\right] \cdot \underbrace{\left(1 - Corr_{x,y}\left[b^{[m]}(x), b^{[m']}(x)\right]\right)}_{\text{Same for all } m' \neq m}$$

From those equations follows:

The expected Loss of the ensemble decreases if we have

- better/less biased base learners.
- more variable base learners.
- less correlation between base learners (this way errors can cancel out)

Bagging works best for unstable/high-variance learners and for base learners with weak correlations, e.g. CART, neural nets.

Not recommended for stable methods like k-NN, LDA, Lin. Regr.

Random Forests

We use CART's as base learners $b^{[m]}(x)$ for our bagging.

For each tree: make the tree large and don't use early stopping or pruning

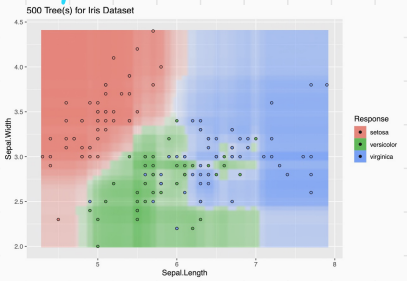
$\Rightarrow$ Increases $\mathbb{V}_{x,y}\left[b^{[m]}(x)\right]$

At each node: randomly select $mtry \leq p$ candidate features to consider for splitting

$\Rightarrow$ Decreases $Corr_{x,y}\left[b^{[m]}(x), b^{[m']}(x)\right]$

- For regression: $mtry = \lfloor \frac{p}{3} \rfloor$
- For classification: $mtry = \lfloor \sqrt{p} \rfloor$

Example



Out-of-Bag Prediction/Error

$$OOB^{[m]} := \left\{ i \in \{1, \dots, n\} \mid (x^{(i)}, y^{(i)}) \text{ is out-of-bag for } b^{[m]} \right\}$$

$$S_{OOB}^{(i)} := \sum_{m=1}^{M} \mathbb{1}(i \in OOB^{[m]}) \quad \text{- amount of trees for which the i-th observation is OOB.}$$

$$\hat{y}_{OOB}^{(i)} = \begin{cases} \frac{1}{S_{OOB}^{(i)}} \sum_{m=1}^{M} \mathbb{1}(i \in OOB^{[m]}) \cdot \hat{y}^{(i)[m]} & \text{, in regression (scalar)} \\ \left[\frac{1}{S_{OOB}^{(i)}} \sum_{m=1}^{M} \mathbb{1}(i \in OOB^{[m]}) \cdot \mathbb{1}\left(\hat{h}^{(i)[m]} = k\right) \right]_{k \in \{1, \dots, g\}} & \text{, in classification (probab. vector)} \end{cases}$$

$$\widehat{err}_{OOB} := \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, \hat{y}_{OOB}^{(i)}\right) \qquad \mathbb{P}(i \in OOB^{[m]}) = (1 - \tfrac{1}{n})^n \to \tfrac{1}{e} \approx 0.37$$

Proximities

We are interested in measuring the similarity of two observations $x^{(i)}$ and $x^{(j)}$.

We define $prox(x^{(i)}, x^{(j)}) = \frac{1}{M} \sum_{m=1}^{M} \mathbb{1}[leaf(i,m) = leaf(j,m)]$. All proximities form an $n \times n$ symmetric matrix.

↑ Leaf in which $x^{(i)}$ lands in Tree m.

We can use Proximities for:

- Imputing missing data:
  1. Replace missing values for a given variable using the median of the non-missing values
  2. Get proximities
  3. Replace missing values in observation $\mathbf{x}^{(i)}$ by a weighted average of non-missing values, with weights proportional to the proximity between observation $\mathbf{x}^{(i)}$ and the observations with the non-missing values

  Steps 2 and 3 are then iterated a few times.
- Locating outliers:
  - An outlier is an observation whose proximities to all other observations are small
  - Measure of outlyingness can be computed for each observation in the training sample
  - If the measure is unusually large, the observation should be carefully inspected
- Identifying mislabeled data:
  - Instances in the training data set are sometimes labeled ambiguously or incorrectly, especially in "manually" created data sets.
  - Proximities can help in finding them: they often show up as outliers in terms of their proximity values.
- Visualizing the forest:
  - The values $1 - prox(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ can be thought of as distances in a high-dimensional space
  - They can be projected onto a low-dimensional space using metric multidimensional scaling (MDS)
  - Metric multidimensional scaling uses eigenvectors of a modified version of the proximity matrix to get scaling coordinates

Variable Importance

It's difficult to evaluate the contributions of different features to the model.

Idea: Measure Performance decrease if a specific feature is removed or made useless.

- Removing feature: (i) Find all nodes $\mathcal{N}$ in $\hat{b}^{[m]}$ that use feature x_j.
  (ii) Compute the improvement that is achieved by x_j in $\mathcal{N}$.
  (iii) Add up all improvements. → That is the importance of x_j.
- Permutation: (i) Perform permutation Ψ_j on feature x_j to distort the feature-target-relation.
  (ii) Calculate the new OOB-Error using the permuted feature $\widehat{err}_{OOB, \Psi_j}$
  (iii) The importance of x_j is measured as $\widehat{err}_{OOB, \Psi_j} - \widehat{err}_{OOB}$