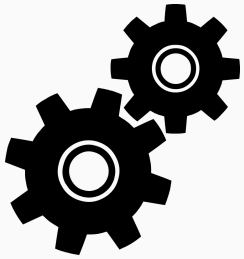
Machine Learning Basics

Prof. Dr. Bernd Bischl

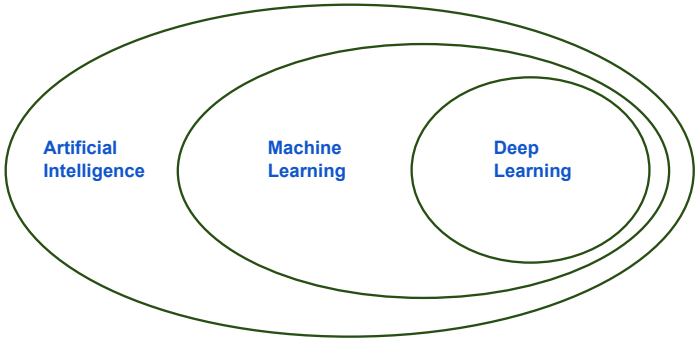
10. September 2018

What is Machine Learning



Machine Learning is a method of teaching computers to make predictions based on some data.

Data Science and Machine Learning
Münchner R Kurse



Machine Learning as Black-Box Modeling

- Many concepts in ML can explained without referring to the inner workings of a certain algorithm or model, especially things like model evaluation and tuning.
- ML also nowadays consists of dozens (or hundreds?) of different modelling techniques, where it is quite unclear which of these are really needed (outside of pure research) and which are really best.
- Understanding model-agnostic techniques is really paramount and can be achieved in a limited amount of time.

ML as Black-Box Modeling

- Really studying the inner workings of each and every ML model can take years. Do we even need to do this at all for some models?
  - No: They exist in software. We can simply try them out, in the best case with an intelligent program that iterates over them and optimizes them.
  - Yes: At least some basic knowledge helps to make right choices. Actually knowing what you are doing is always good, also outside of science. And often stuff goes wrong, then understanding helps, too.

ML as Black-Box Modeling

- In the follwoing slides we will go through really fundamental terms and concepts in ML, that are relevant for everything that comes next.
- We will also learn a couple of extremely simple models to obtain a basic understanding.
- More complex stuff comes later.

ML as Black-Box Modeling

Imagine you want to investigate how salary and workplace conditions effect productivity of employees. Therefore, you collect data about worked minutes per week (productivity), how many persons work at the same office of the employee and his salary.

Data, Target and Features

		Features x			
Worked Minutes Week (Target Variable)	y	People in Office (Feature 1)	x_1	Salary (Feature 2)	x_2
2220	$y^{(1)}$	4	$x_1^{(1)}$	4300 €	$x_2^{(1)}$
1800	$y^{(2)}$	12	$x_1^{(2)}$	2700 €	$x_2^{(2)}$
1920	$y^{(3)}$	5	$x_1^{(3)}$	3100 €	$x_2^{(3)}$

$n = 3$

$p = 2$

The whole data set is expressed by

$$\mathcal{D} = \left\{ \left(x^{(1)}, y^{(1)} \right), \ldots, \left(x^{(n)}, y^{(n)} \right) \right\}$$

with the i-th observation $\left(x^{(i)}, y^{(i)} \right)$.

Target and Features Relationship

- For our observed data we know which outcome is produced:

y	x_1	x_2
2200	4	4300 €
1800	12	2700 €
1920	15	3100 €

Already seen Data

Target and Features Relationship

- For new employees we can just observe the features:

y	x_1	x_2	
2200	4	4300 €	Already seen Data
1800	12	2700 €	
1920	15	3100 €	
???	6	3300 €	New Data
???	5	3100 €	

⇒ The goal is to predict the target variable for **unseen new data** by using a **model** trained on the already seen **train data**.

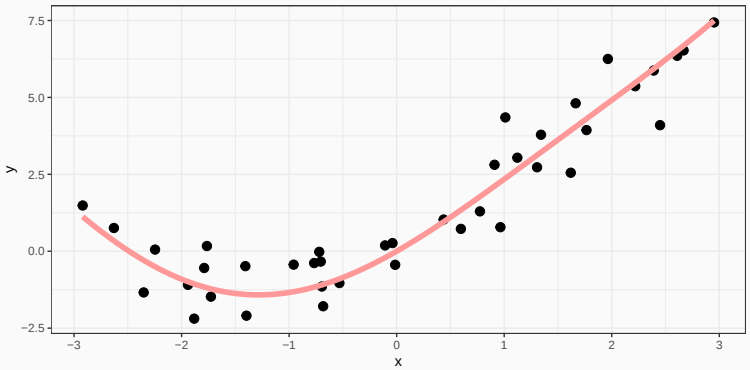
Supervised Learning Task

- **Regression task** if we have to predict a numeric target variable, e.g., the minutes an employee works per week.
- **Classification task** if we have to predict a categorical target state, e.g., if an employee is happy with his job or not.

Regression Task

Münchner R Kurse

- **Goal**: Predict a continuous output
- y is metric variable (with values in $\mathbb{R}$)
- Regression model can be constructed by different methods, e.g., linear regression, trees or splines



Target and Features Relationship

Münchner R Kurse

y	Functional Relationship	x_1	x_2	
2200		4	4300 €	Already seen Data
1800		12	2700 €	
1920	f	15	3100 €	
???		6	3300 €	New Data
???		5	3100 €	

Target and Features Relationship

Münchner R Kurse

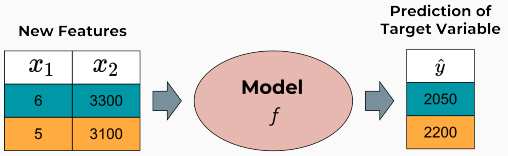
- In ML, we want to be "lazy". We do not want to specify f manually.
- We want to learn it **automatically from labeled data**.
- Later we will see that we do have to specify something, like f's functional form and other stuff.
- Mathematically, we face a problem of function approximation: search for an f, such that, for all points in the training data, and also all newly observed points

- We call this **supervised learning**.

What is a Model?

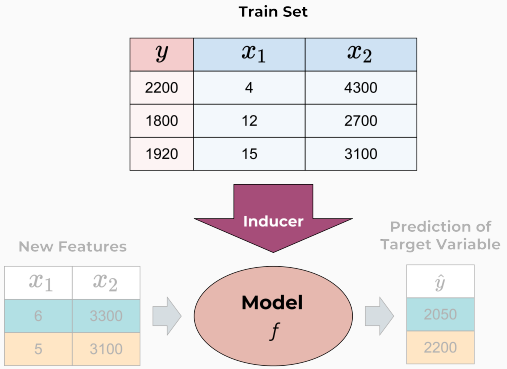
Münchner R Kurse

A model takes the features of new observations and produces a prediction $\hat{y}$ of our target variable y:



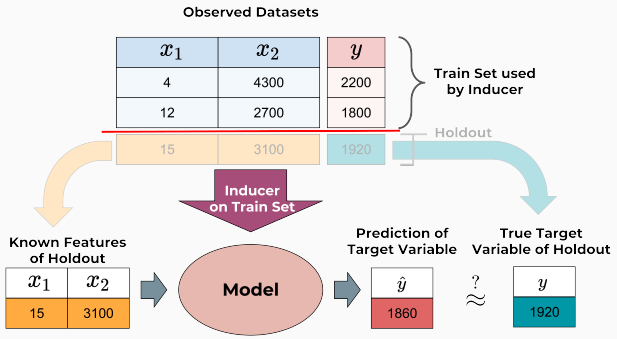
What is an Inducer?

The **inducer** (learner, algorithm) takes our labeled data set (**training set**) and produces a model (which again is a function):



How to Evaluate Models

- Simply compare predictions from model with truth:



- We will learn much more about this later!

Inducer Decomposition

Nearly all ML supervised learning training algorithms can be described by three components:

Learning = Representation + Evaluation + Optimization

- **Representation / Hypothesis Space:** Defines functional structures of f we can learn.
- **Evaluation:** How well does a certain hypothesis score on a given data set? Allows us to choose better candidates over worse ones.
- **Optimization:** How do we search the hypothesis space? Guided by the evaluation metric.

Inducer Decomposition

- All of these components represent important choices in ML which can have drastic effects:
- If we make smart choices here, we can tailor our inducer to our needs - but that usually requires quite a lot of experience and deeper insights into ML.

Inducer Decomposition

Münchner R Kurse

Representation	Evaluation	Optimization
Instances / Neighbours	Squared error	Gradient descent
Linear functions	Likelihood	Stochastic gradient descent
Decision trees	Information gain	Quadratic programming
Set of rules	K-L divergence	Greedy optimization
Neural networks		Combinatorial optimization
Graphical models		

Note: What is on the same line above does not belong together!

Loss Minimization

Münchner R Kurse

- Let $\mathbb{P}_{xy}$ be the joint distribution of x and y. This defines all aspects of the generating process where our data comes from. Sadly, we usually do not $\mathbb{P}_{xy}$.
- The goodness of predictions – per point – of a model $y = f(x|\theta)$ – w.r.t. a parameter vector θ – is measured by a loss function, e.g. squared error:

$$L(y, f(x))) = (y - f(x|\theta))^2$$

- Obvious aim: minimization of prediction error over all θ with respect to data distribution $\mathbb{P}_{xy}$. This error is called the (true / theoretical) risk.

Supervised Learning

Risk Minimization

Münchner R Kurse

- As we usually do not know / cannot compute the theoretical risk, we approximate this on our empirical data. This is called empirical risk:

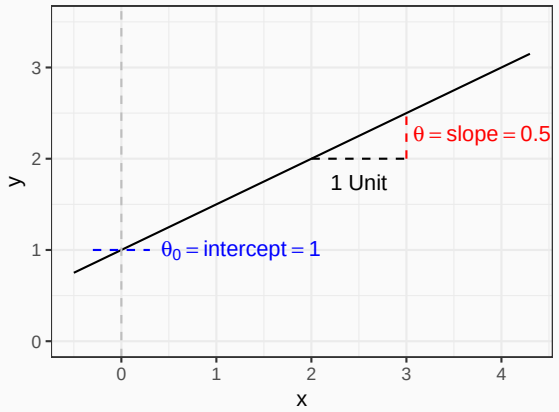
$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right)$$

- Then minimize it over θ. This is called empirical risk minimization:

$$\arg\min_{\theta} \mathcal{R}_{\text{emp}}(\theta) = \arg\min_{\theta} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right)$$

The Linear Model

Straight Line / Univariate Linear Model



$$y = \theta_0 + \theta \cdot x$$

Linear Regression: Representation

We want to learn a numerical target variable, by a linear transformation of the features. So with $\theta \in \mathbb{R}^p$ this mapping can be written:

$$y = f(x) = \theta_0 + \theta^T x$$

This restricts the hypothesis space H to all linear functions in θ:

$$H = \{\theta_0 + \theta^T x \mid (\theta_0, \theta) \in \mathbb{R}^{p+1}\}$$

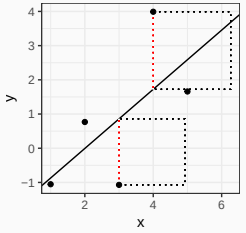
Given observed labeled data $\mathcal{D}$, how to find (θ, θ_0)? This is learning or parameter estimation, the inducer does exactly this.

NB: We assume here and from now on that θ_0 is included in θ.

Linear Regression: Evaluation

We now measure training error as sum-of-squared errors. This is also called the L2 loss, or L2 risk:

$$\mathcal{R}_{\text{emp}}(\theta) = \text{SSE}(\theta) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right) = \sum_{i=1}^{n} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$



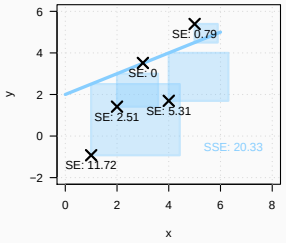
Optimizing the squared error is computationally much simpler than measuring the absolute differences (this would be called L1 loss).

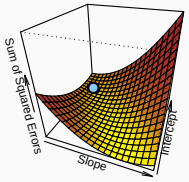
Linear Model: Optimization

Münchner R Kurse

We want to find the parameters of the LM / an element of the hypothesis space H that best suits the data. So we evaluate different candidates for θ.

- A first (random) try yields an SSE of 20.33 (**Evaluation**).



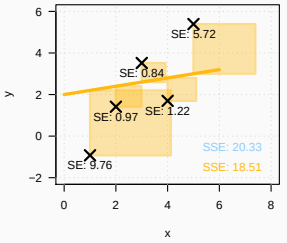


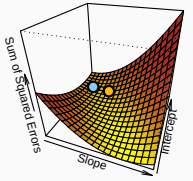
Linear Model: Optimization

Münchner R Kurse

We want to find the parameters of the LM / an element of the hypothesis space H that best suits the data. So we evaluate different candidates for θ.

- Another line yields an lower SSE of 18.51 (**Evaluation**). Therefore, this one is better in terms of empirical risk.





Linear Model: Optimization

Münchner R Kurse

Instead of guessing parameters we could also use gradient descent to iteratively find the optimal θ. Here, we can even find the optimal value analytically:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{R}_{\text{emp}}(\theta) = \arg\min_{\theta} \|y - X\theta\|_2^2$$

X is the $n \times (p+1)$-feature-data-matrix, also called design matrix.

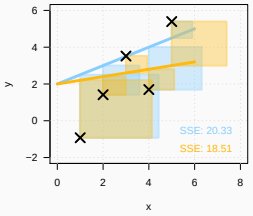
This yields the so called normal equations for the LM:

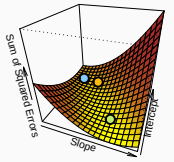
$$\frac{\delta}{\delta\theta}\mathcal{R}_{\text{emp}}(\theta) = 0 \qquad \Rightarrow \qquad \hat{\theta} = \left(X^T X\right)^{-1} X^T y$$

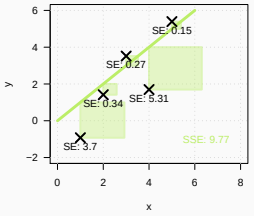
Linear Model: Optimization

Münchner R Kurse

The optimal line has an SSE of 9.77 (**Evaluation**)





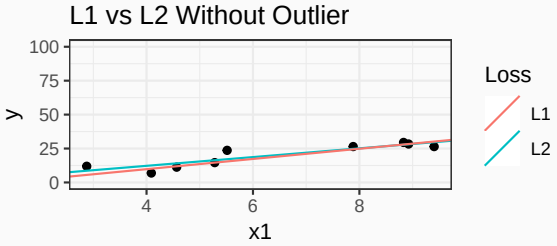


Example: Linear Regr. With L1 vs L2 Loss

We could also minimize the L1 loss. This changes the evaluation and optimization step:

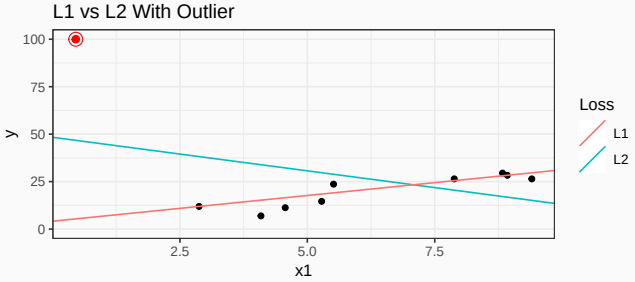
$$\mathcal{R}_{\text{emp}}(\theta) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(x^{(i)}|\theta\right)\right) = \sum_{i=1}^{n} \left|y^{(i)} - \theta^T x^{(i)}\right| \qquad \text{(Evaluation)}$$

Much harder to optimize, but model is less sensitive to outliers.



Example: Linear Regr. With L1 vs L2 Loss

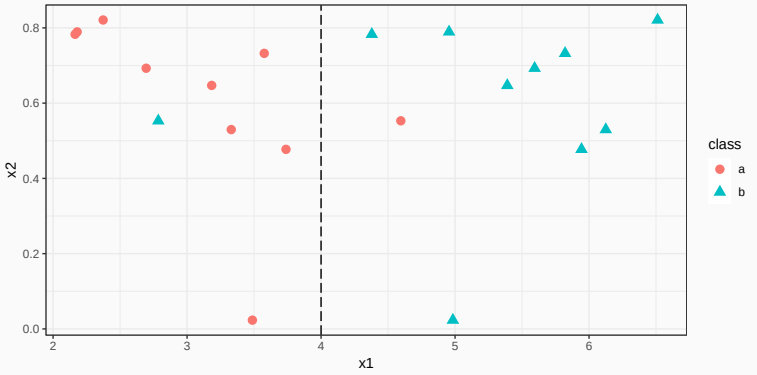
Adding an outlier (highlighted red) pulls the line fitted with L2 into the direction of the outlier:



Classification

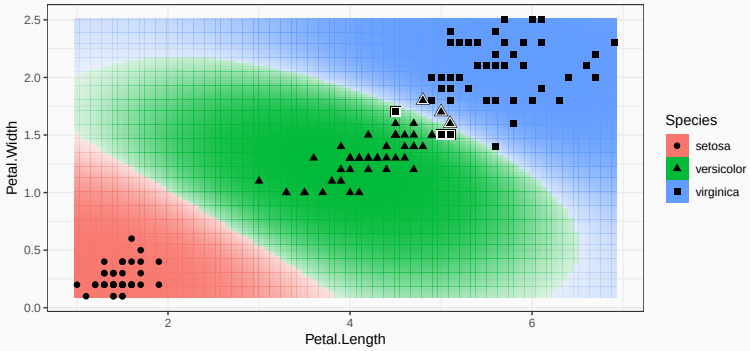
Binary Classification Task

- y is a categorical variable (with two values)
- E.g., sick/healthy, or credit/no credit
- **Goal**: Predict a class (or membership probabilities)



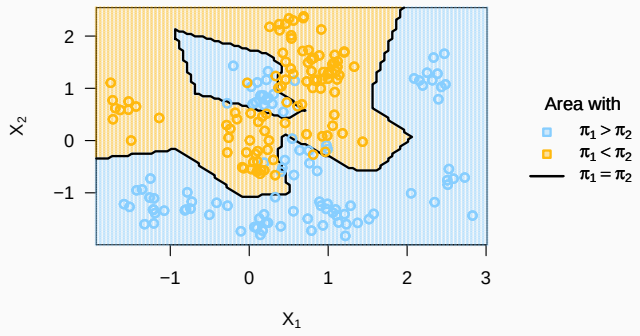
Multiclass Classification Task

- y is a categorical variable with > 2 unordered values
- Each instance belongs to only one class
- **Goal**: Predict a class (or membership probabilities)



Classification Task: Decision Boundary

- Area in feature space where there is no unique best class with maximum posterior probability are equal, e.g., for binary classification $\pi(x)_1 = \pi(x)_2$
- Separates areas in which a specific class is estimated / chosen.



Classification Task: Posterior Probabilities

- Often estimating class probabilities for observations is more informative than predicting labels.
- Posterior probabilities are the probability of a class k occuring, given the feature x

$$\pi_k(x) = P(Y = k|x)$$

- Many models can directly output and optimize for these probabilities.
- Class estimation for x via $\arg\max_k \pi_k(x)$ - just take most likely class.
- We will talk much more about this in ROC analysis.

K-Nearest Neighbors

Supervised: Geometric Approach

Idea: Relate outputs of observations x and $\tilde{x}$ to each other, if their distance in input space is not large.

E.g.: $\tilde{x}$ should have the same label as x, as the feature values of $\tilde{x}$ are nearly the same as the values of x, i.e. $d(x, \tilde{x})$ is small.

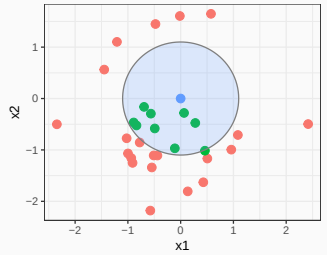
Popular distance for numerical data: **Euclidean distance**

$$d_{\text{Euclidean}}(x, \tilde{x}) = \sqrt{\sum_{j=1}^{p} (x_j - \tilde{x}_j)^2}$$

It is based on a special case of the L_q-norm:
$||x||_q = (|x_1|^q + ... + |x_q|^q)^{\frac{1}{p}}$ with $q = 2$.

k-Nearest-Neighbors

- k-NN is a non-parametric method for regression and classification
- Models predictions $\hat{y}$ for x by looking at $\left(x^{(i)}, y^{(i)}\right)$ near x
- Closeness implies distance measure (usually Euclidean)
- $N_k(x)$ is called the *neighborhood* of x, if it consists of the k-closest points $x^{(i)}$ to x in the training sample



k-Nearest-Neighbors

Predictions:

- For regression, average over neighbors' outputs:

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

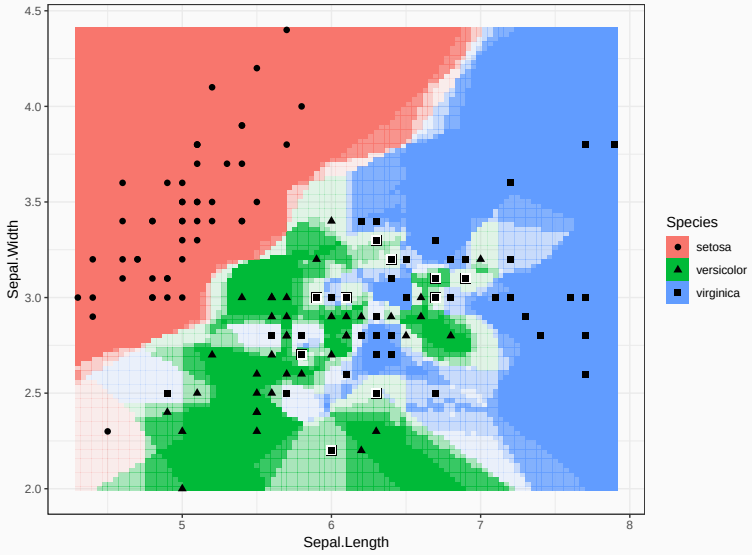
- For classification: majority vote over neighbors' labels:

$$\hat{y} = \arg\max_{l} \sum_{x^{(i)} \in N_k(x)} \mathbb{I}\left(y^{(i)} = l\right)$$

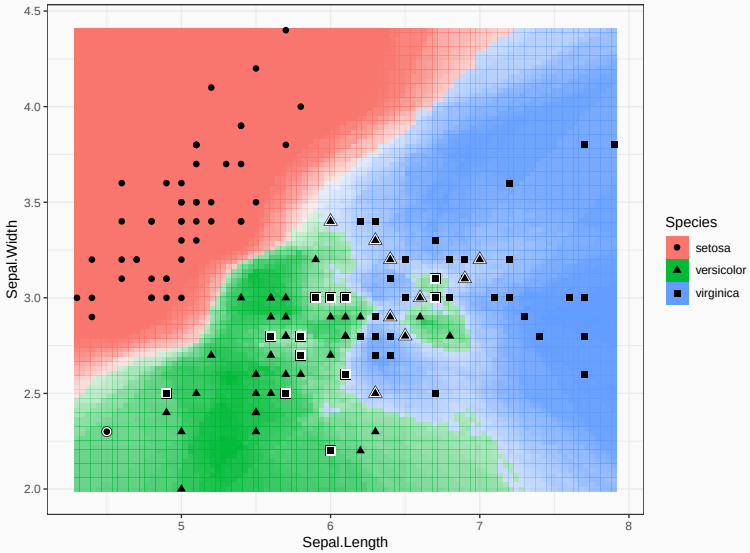
Posterior probabilities can be estimated with:

$$\hat{\pi}_l(x) = \frac{1}{k} \sum_{x^{(i)} \in N_k(x)} \mathbb{I}\left(y^{(i)} = l\right)$$

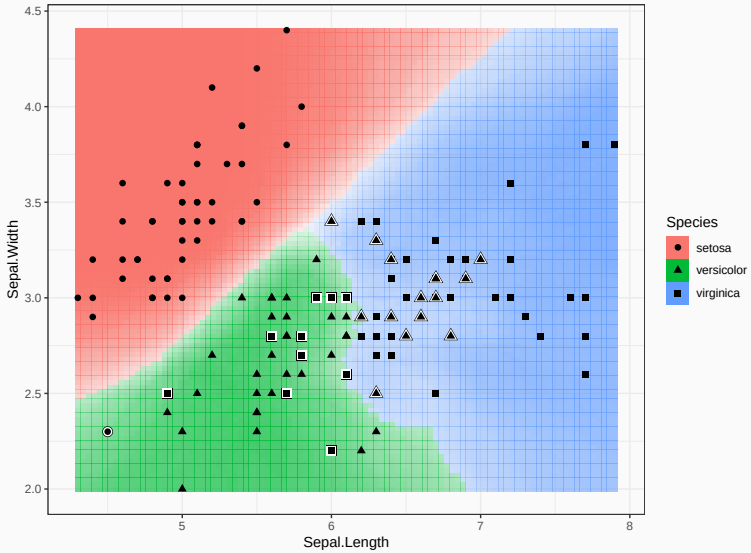
k-Nearest-Neighbors: k = 3



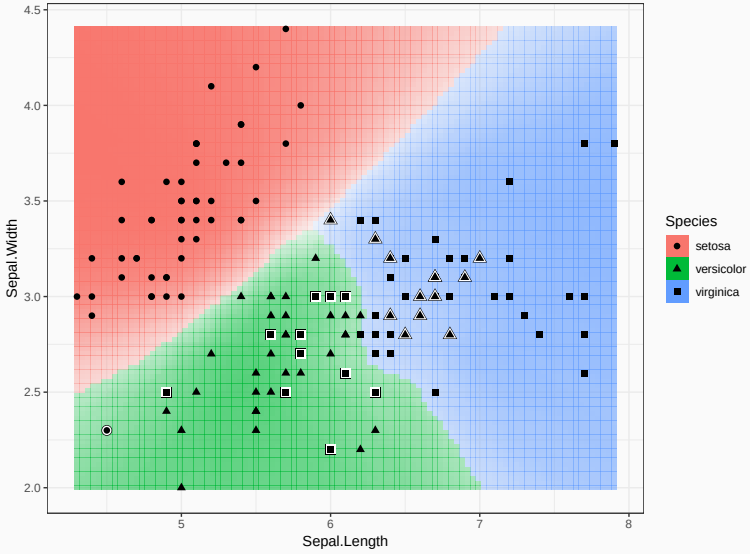
k-Nearest-Neighbors: k = 10



k-Nearest-Neighbors: k = 30



k-Nearest-Neighbors: k = 50



k-Nearest-Neighbors

- No assumptions about the underlying data distribution.
- No training-step and is a very local model.
- Many more parameters than the simpler linear model.
- The smaller k, the less stable, less smooth and more "wiggly" the decision boundary becomes.
- Accuracy of k-NN can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance.

ML vs. Statistics

Modeling: Two Cultures

Typical assumptions and restrictions

- Stochastic model for the data-generating process
- Linearity (e.g. linear predictor)
- Manual specification of interactions

Problems

- Conclusions about model, not about nature
- Assumptions often violated
- Often no test-set evaluation
- Often irrelevant (?) theory
- Focus not on prediction, can fail in areas like image and speech recognition

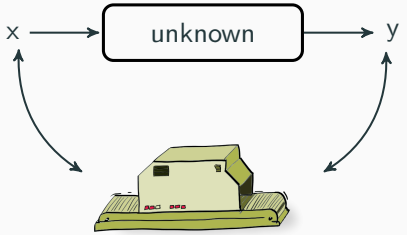
Modeling: Two Cultures

Statistics, the Data Modeling Culture

- Focuses on the modeling of data to get a better understanding about the relationship between covariates and outcome
- Usually we have to assume how the data generating process looks as a stochastic model

Modeling: Two Cultures

Machine Learning, the Algorithmic Modeling Culture



algorithm

Find a function $f(x)$ that minimizes the loss: $L(y, f(x))$

Modeling: Two Cultures

- In the "algorithmic modeling culture", the true mechanism is treated as unknown
- We do not strive to find the true data-generating mechanism, but to find a model that works as good as possible - in terms of prediction.
- Modeling is reduced to a mere problem of function approximation: Find model that fits data best.

Modeling: Two Cultures

Summary

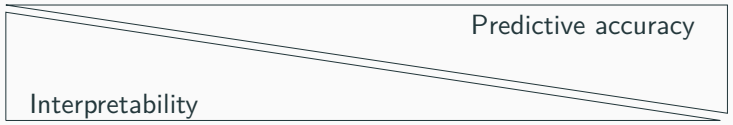
The data modeling culture tries to find the true data-generating mechanism, the algorithmic modeling culture tries to imitate the true mechanism as good as possible.

Rashomon Effect

(Often) Many different models describe a situation equally accurate which makes it difficult to find the true mechanism in the data modeling cultures.

Modeling: Two Cultures

There is a trade-off between interpretability and predictive accuracy: the models that are good in prediction are often complex and models that are easy to interpret are often bad predictors.



Tree

Logistic Regression

...

Random Forest

Neural networks

...

Modeling: Two Cultures

Machine Learning	Statistics
Feature,Attribute	Covariate
Minimizing loss	Maximizing likelihood
Learning	Fitting/Estimation
Weights	Parameter/Coefficient
Bias term	Intercept
Hypothesis	Model
Example/Instance	Observation
Supervised Learning	Regression/Classification
Label	Response
Data mining (good)	Data mining (bad)
Log. regr. is classification	Log. regr. is regression

Model Evaluation

Prof. Dr. Bernd Bischl

10. September 2018

Performance Metrics

Performance Measures

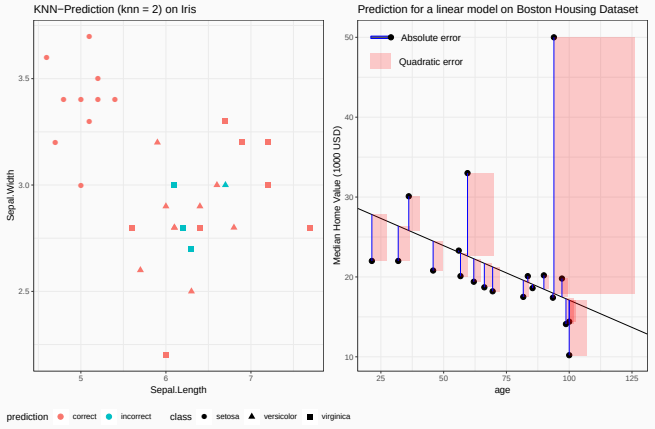
Münchner R Kurse

"The generalization perormance of a learning method relates to its prediction capability on independent test data. Assesment of this performance is extremely important in practice, since it guides the choice of learning method or model, and gives us a measure of the quality of the ultimately chosen model."

Hastie et al. - The Elements of Statistical Learning

Performance Measures

Münchner R Kurse



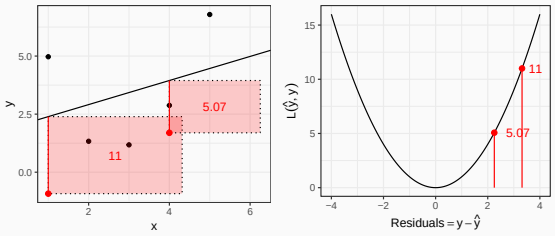
To compare the performance of two ML models on a single observation or a new data set, we would ideally like to boil them down to a **single** numeric measure of the performance.

Regression: MSE

The **Mean Squared Error** compares the mean of the squared distances between the target variable y and the predicted target $\hat{y}$.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y})^2 \in [0; \infty]$$

Single observations with a large prediction error heavily influence the **MSE**, as they enter quadratically.

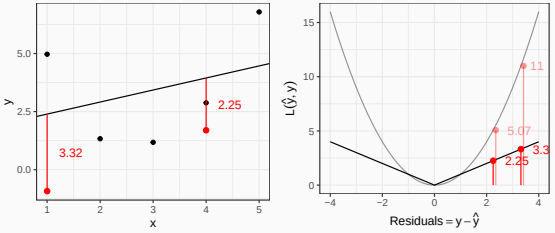


Regression: MAE

A more robust (but not neccessarily better) way to compute a performance measure is the **Mean Absolute Error**:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - \hat{y}| \in [0; \infty]$$

The absolute error is more robust. On the other hand, the absolute error is not differentiable at 0.



Regression: R^2

Another often used measure is R^2. It measures the *fraction of variance explained* by the model.

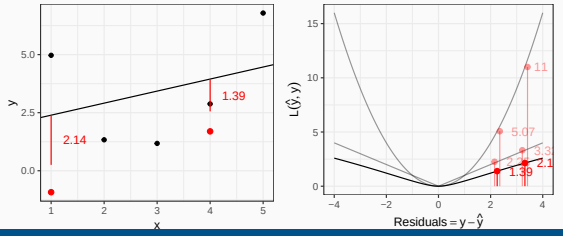
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y^{(i)} - \hat{y})^2}{\sum_{i=1}^{n}(y^{(i)} - \bar{y})^2} = \frac{SSE_{Model}}{SSE_{Intercept}}$$

- If R^2 is measured on the training data: $R^2 \in [0; 1]$.
- If R^2 is measured on held out data: $R^2 \in [-\infty; 1]$.

Regression: Defining a custom loss

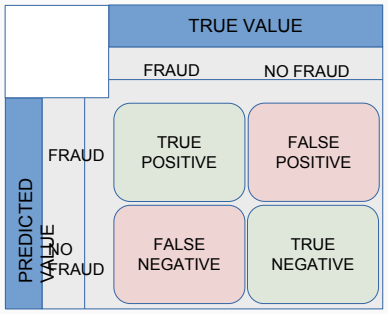
Assume a use case, where the target variable can have a wide range of values across different orders of magnitude. A possible solution would be to use a loss functions that allows for better model evaluation and comparison. The **Mean Squared Logarithmic Absolute Error** is not strongly influenced by large values due to the logarithm.

$$\frac{1}{n}\sum_{i=1}^{n}(\log(|y^{(i)} - \hat{y}| + 1))^2$$



Performance Measures - Classification

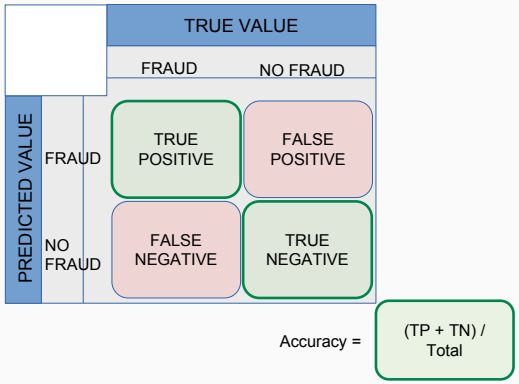
The Confusion matrix is an intuitive metrics used for establishing the correctness and accuracy of a model. It is used for classification problems where the output can be of two or more types of classes.



Classification: Accuracy / Classification Error

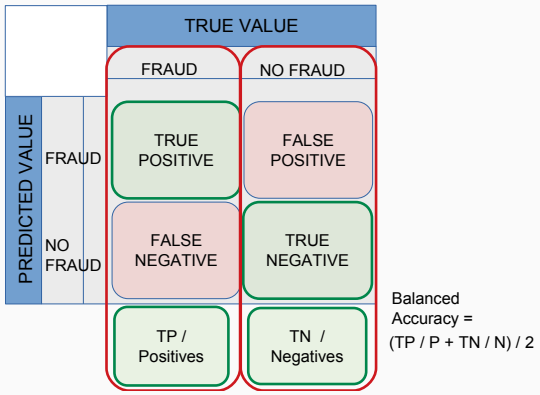
The Confusion matrix is not a performance measure as such, but most performance metrics for binary classification are based on it.

Accuracy; Classification Error = 1 - Accuracy



Classification: Balanced Accuracy / Error

Problem: Accuracy is sensitive for imbalanced data situations.
Solution: Compute the accuracy for each class label



Classification: Brier Score

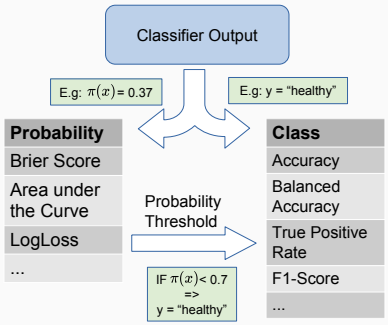
Measures squared distances of probabilities from the true class labels:

$$BS = \frac{1}{n} \sum_{i=1}^{n} \left(\pi^{(i)}(x) - y^{(i)} \right)^2$$

- Above is the usual definition for binary case, $y^{(i)}$ must be coded as 0 and 1.
- Measures how well calibrated probabilities are.

Threshold

Thresholding flexibly converts measured probabilities to labels.



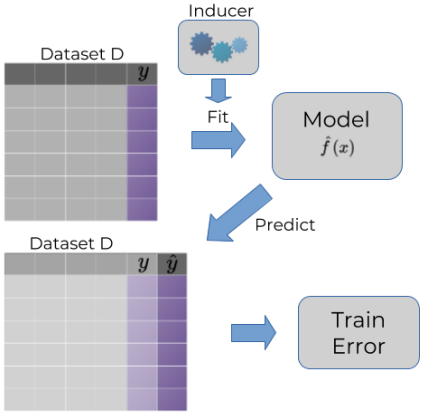
List of Classification Performance Measures

Classification	Explanation
Accuracy	Fraction of correct classifications
Balanced Accuracy	Fraction of correct classifications in each class
Recall	Fraction of positives a classifier captures
Precision	Fraction of positives in instances predicted as positive
F1-Score	Tradeoff between precision and recall
AUC	Measures calibration of predicted probabilities
Brier Score	Squared difference between predicted probability and true label
LogLoss	Emphasizes errors for predicted probabilities close to 0 and 1

Train and Test Error

Training Error

The *training error* (also called apparent error or resubstitution error) is estimated by the average error over the training set $\mathcal{D}_{\text{train}}$:



Training Error

- The training error is usually a very unreliable and overly optimistic estimator of future performance.
- Goodness-of-fit measures like R^2, likelihood, AIC, BIC, deviance are all based on the training error.

Example: Polynomial Regression

Assume we have a single, univariate x and that y can be approximated by a dth-order polynomial

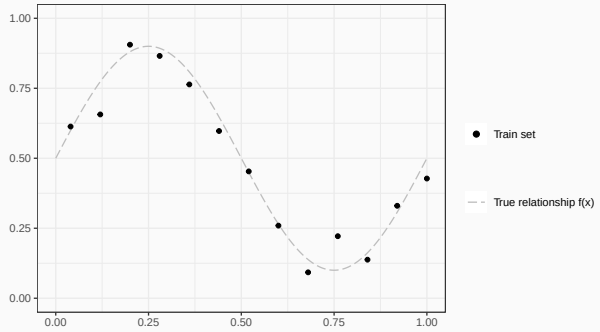
$$f(x|\theta) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^{d} \theta_j x^j.$$

θ_j are the coefficients and d is called the degree.

This is still linear regression / a linear model, and for a fixed d, the θ can easily be obtained through the normal equations.

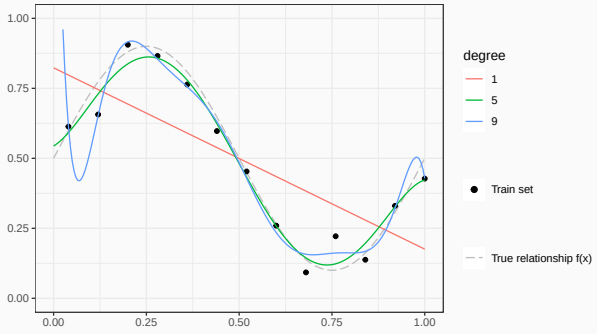
Example: Polynomial Regression

True relationship is $f(x) = 0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$



Example: Polynomial Regression

Models of different *complexity*, i.e., of different orders of the polynomial are fitted. How should we choose d?



Example: Polynomial regression

The performance of the models on the training data can be measured by calculating the *training error* according to the mean squared error (MSE) criterion:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

d = 1: 0.03, d = 5: 0.002, d = 9: 0.001

Simply using the training error seems to be a bad idea.

Training Error Problems

- Extend any ML training in the following way: On top of normal fitting, we also store $\mathcal{D}_{\text{train}}$. During prediction, we first check whether x is already stored in this set. If so, we replicate its label. The training error of such an (unreasonable) procedure will always be zero.
- The training error of 1-NN is always zero.
- The training error of an interpolating spline in regression is always zero.

Training Error Problems

- For models of severely restricted capacity, and given enough data, the training error might provide reliable information. E.g. consider a linear model in 5d, with 10.000 training points. But: What happens if we have less data? And p becomes larger? Can you precisely define where the training error becomes unreliable?

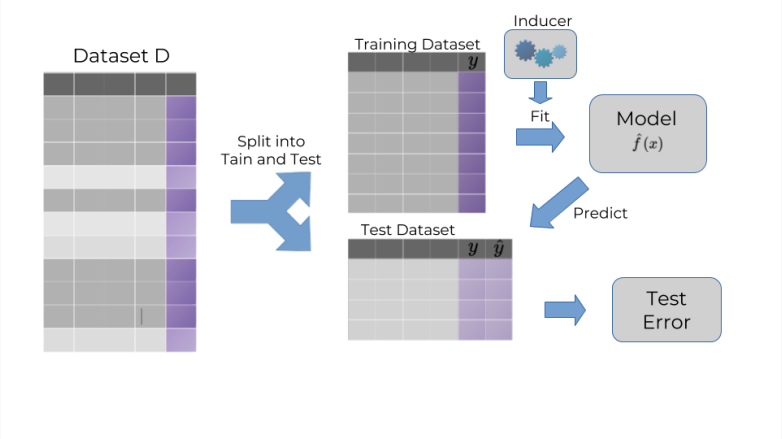
Test Error and Hold-Out Splitting

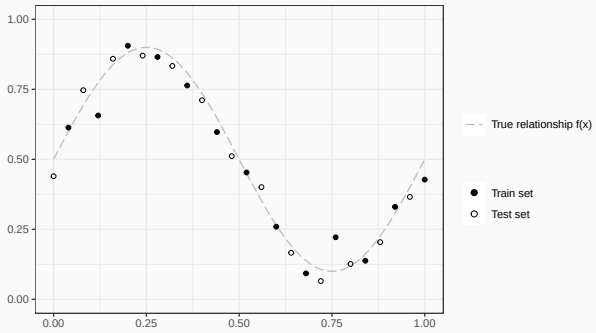
To reliably assess a model, we need to define

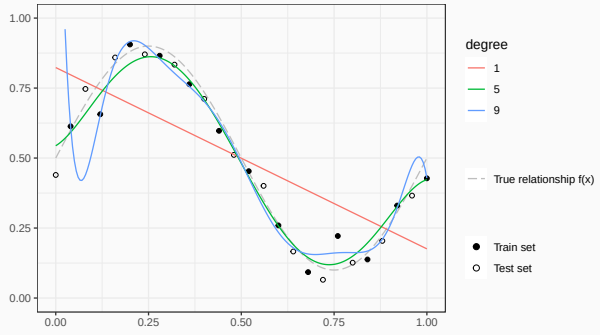
- how to simulate the scenario of "new unseen data" and
- how to estimate the generalization error.

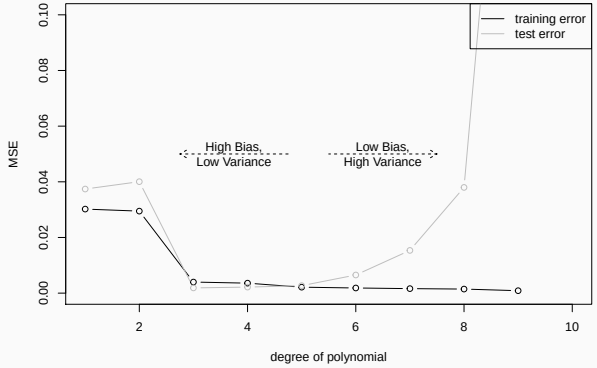
Hold-out splitting and evaluation

- Straightforward idea: To measure performance, let's simulate how our model will be applied on new, unseen data
- So, to evaluate a given model: predict and evaluate only on data not used during training
- For a given set $\mathcal{D}$, we have to preserve some data for testing that we cannot use for training, hence we need to define a training set $\mathcal{D}_{\text{train}}$ and a test set $\mathcal{D}_{\text{test}}$, e.g. by randomly partitioning the original $\mathcal{D}$









Test error is best for $d = 3$

Training vs. Test Error

Münchner R Kurse

The training error

- is an over-optimistic (biased) estimator as the performance is measured on the same data the learned prediction function $\hat{f}_{\mathcal{D}_{\text{train}}}(x)$ was trained for.
- decreases with smaller training set size as it is easier for the model to learn the underlying structure in the training set perfectly.
- decreases with increasing model complexity as the model is able to learn more complex structures.

Training vs. Test Error

Münchner R Kurse

The test error

- will typically decrease when the training set increases as the model generalizes better with more data.
- will have higher variance with decreasing test set size.
- will have higher variance with increasing model complexity.

Bias-Variance of Hold-Out

Münchner R Kurse

- If the size of our initial, complete data set $\mathcal{D}$ is limited, single train-test splits can be problematic.
- The smaller our single test set is, the higher the variance of our estimated performance error (e.g., if we test on one observation, in the extreme case). But note that by just making the test set smaller, we do not introduce any bias, as we simply average losses on i.i.d. observations from $\mathbb{P}_{xy}$.
- The smaller our training set becomes, the more pessimistic bias we introduce into the model. Note that if $|D| = n$, our aim is to estimate the performance of a model fitted on n observations (as this is what we will do in the end). If we fit on less data during evaluation, our model will learn less, and perform worse. Very small training sets will also increase variance a bit.

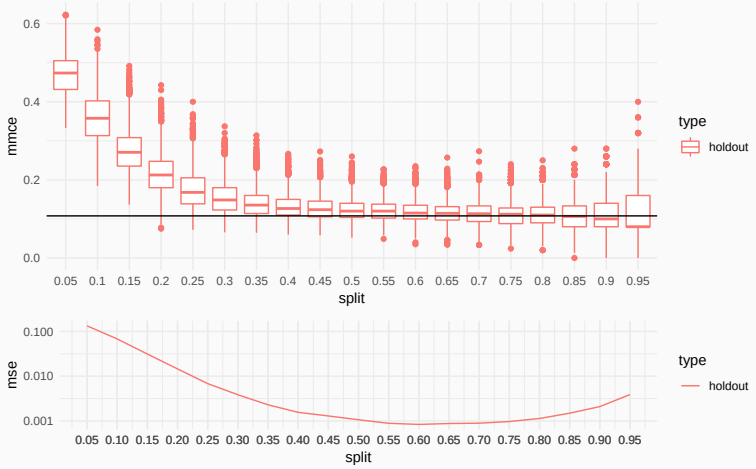
Bias-Variance of Hold-Out - Experiment

Münchner R Kurse

- Data: simulate spiral data (sd $= 0.1$) from `mlbench`
- Learner: CART (`classif.rpart` from `mlr`)
- Goal: estimate real performance of a model with $|\mathcal{D}_{\text{train}}| = 500$
- Analyse different types of hold-out and subsampling ($=$ repeated hold-out)
- Sample $\mathcal{D}$ with $|\mathcal{D}| = 500$ and use split-rate $s \in \{0.05, 0.1, ..., 0.95\}$ for training with $|\mathcal{D}_{\text{train}}| = s \cdot 500$
- Estimate performance on $\mathcal{D}_{\text{test}}$ with $|\mathcal{D}_{\text{test}}| = 500 \cdot (1 - s)$
- Repeat the sampling of $\mathcal{D}$ 50 times and the splitting with s 50 times

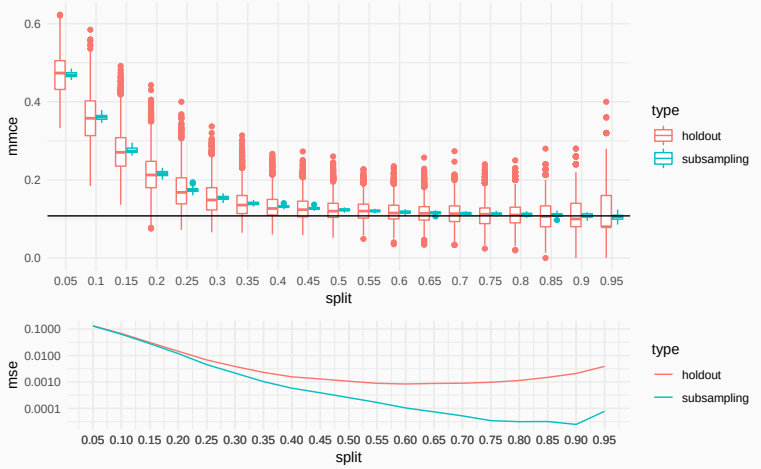
Bias-Variance of Hold-Out - Experiment

MSE of the estimator in relation to the true error rate:



Bias-Variance of Hold-Out - Experiment

MSE of the estimator in relation to the true error rate:

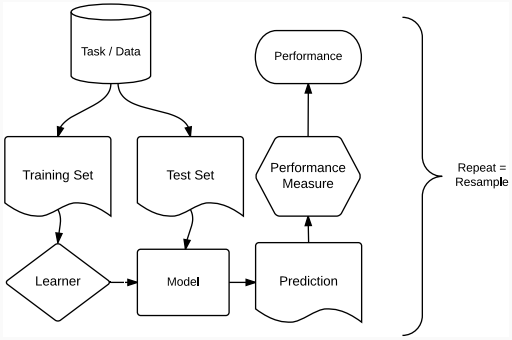


Resampling

- We need to construct a better performance estimator through *resampling*, that uses the data more efficiently.
- All resampling variants operate similar: The data set is split repeatedly into training and tests sets, and we later aggregate (e.g. average) the results.
- The usual trick is to make training sets quite larger (to keep the pessimistic bias small), and to handle the variance introduced by smaller test sets through many repetitions and averaging of results.

Resampling

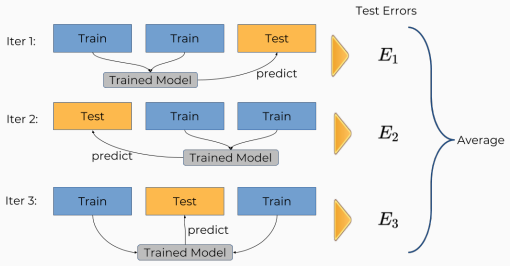
- Aim: Assess the performance of learning algorithm.
- Uses the data more efficiently then simple train-test.
- Repeatedly split in train and test, then aggregate results.



Cross-Validation

- Split the data into k roughly equally-sized partitions.
- Use each part once as test set and joint $k-1$ other as train.
- Obtain k test errors and average.

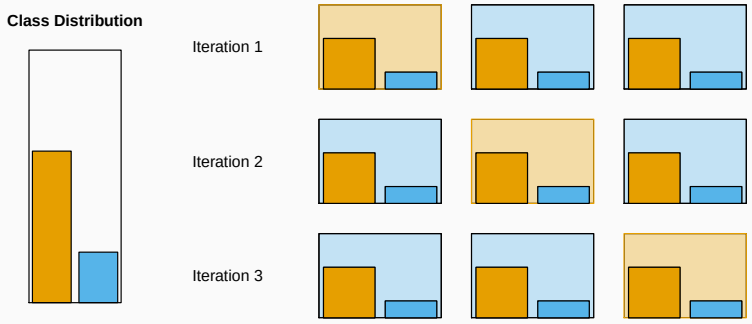
Example: 3-fold cross-validation:



Cross-Validation - Stratification

Stratification tries to keep the distribution of the target class (or any specific categorical feature of interest) in each fold.

Example of stratified 3-fold Cross-Validation:



Cross-Validation - Comments

- $k = n$ is known as leave-one-out (LOO) cross-validation or jackknife.
- The performance estimates for each fold are NOT independent, because of the structured overlap of the training sets. Hence, the variance of the estimator increases again for very large k (close to LOO), when training sets nearly completely overlap.
- LOO is nearly unbiased, but has high variance.
- Repeated k-fold CV (multiple random partitions) can improve error estimation for small sample size.

Subsampling

- Use a randomly selected proportion of the data for training, the other for test. Repeat k times.
- $k = 30$ or $k = 100$ are popular choices, which quickly can become computationally demanding.
- Holdout: Subsampling with $k = 1$.
- Bootstrapping: Draw observations with replacement.

Summary of concepts

- In ML we fit, at the end, a model on all our given data.
- Problem: We need to know how well this model performs in the future. But no data now is left to reliably do this. But we really need this.
- In order to approximate this, we do the next best thing. We estimate how well the inducer works when it sees nearly n points from the same data distribution.
- Holdout, CV, resampling estimate exactly this number. The "pessimistic bias" refers to when use much less data in fitting than n. Then we "hurt" our inducer unfairly.

Summary of concepts

- Strictly speaking, resampling only produces one number, the performance estimator. It does NOT produce models, paramaters, etc. These are intermediate results and discarded.
- The model and parameters are obtained when we fit the inducer finally on the complete data.
- This is a bit weird and complicated, but we have to live with this.

Comments

- 5CV or 10CV have become standard, 10-times repeated 10CV for small sample sizes, but THINK about whether that makes sense in your application.
- Do not use Hold-Out, CV with few iterations, or subsampling with a low subsampling rate for small samples, since this can cause the estimator to be extremely biased, with large variance.
- A $\mathcal{D}$ with $|\mathcal{D}| = 100.000$ can have small sample size properties if one class has only 100 observations ...

Overfitting

Overfitting

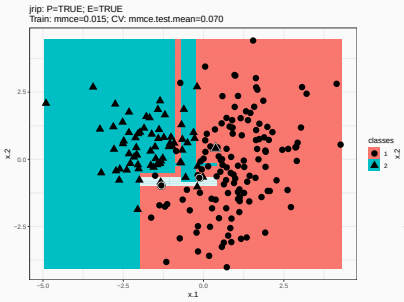
Münchner R Kurse

- Learner finds a pattern in the data that is not actually true in the real world: *overfits* the data
- Every powerful learner can "hallucinate" patterns
- Happens when you have too many hypotheses and not enough data to tell them apart
- The more data, the more "bad" hypotheses are eliminated
- If the hypothesis space is not constrained, there may never be enough data
- There is often a parameter that allows you to constrain (*regularize*) the learner

Overfitting

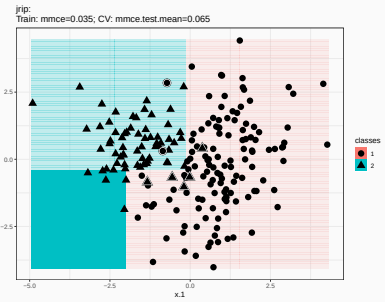
Münchner R Kurse

Overfitting learner



Better training set performance (seen examples)

Non-overfitting learner

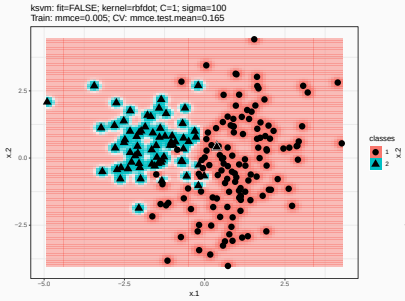


Better test set performance (unseen examples)

Overfitting

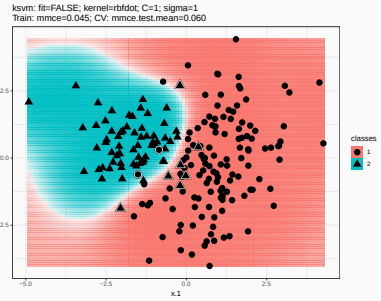
Münchner R Kurse

Overfitting rule learner



Better training set performance (seen examples)

Non-overfitting rule learner



Better test set performance (unseen examples)

Overfitting and Noise

Münchner R Kurse

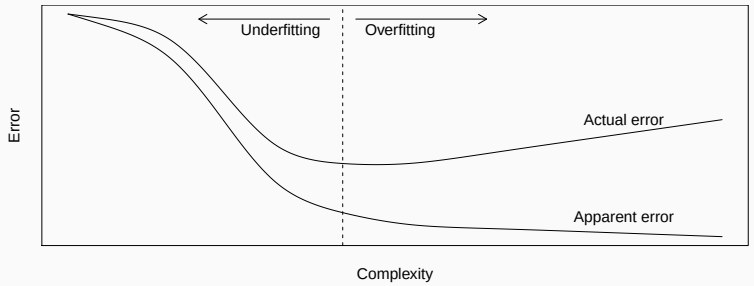
- Overfitting is seriously exacerbated by *noise* (errors in the training data)
- An unconstrained learner will start to model that noise
- It can also arise when relevant features are missing in the data
- In general it's better to make some mistakes on training data ("ignore some observations") than trying to get all correct

Avoiding Overfitting

- Some learner can do "early stopping" before perfectly fitting (i.e., overfitting) the training data
- In general, prefer simpler hypotheses altogether when they work well (e.g. regularization, pruning)

Trade-Off Between Generalization Error and Complexity

Apparent error (on the training data) and real error (prediction error on new data) evolve in the opposite direction with increasing complexity:



⇒ Optimization regarding the model complexity is desirable: Find the right amount of complexity for the given amount of data where generalization error becomes minimal.

Triple Trade-Off

In all learning algorithms that are trained from data, there is a trade-off between three factors:

- The complexity of the hypothesis we fit to the training data
- The amount of training data (in terms of both instances and informative features)
- The generalization error on new examples

If the capacity of the learning algorithm is large enough to a) approximate the data generating process and b) exploit the information contained in the data, the generalization error will decrease as the amount of training data increases.

For a fixed size of training data, the generalization error decreases first and then starts to increase (overfitting) as the complexity of the hypothesis space H increases.

Trees and Random Forest

Prof. Dr. Bernd Bischl

10. September 2018

Trees

Introduction to CART

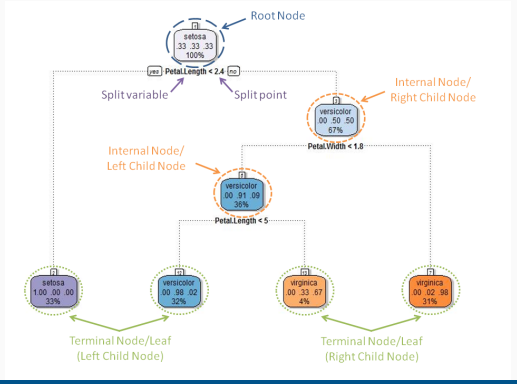
Can be used for classification, regression (and much more!)

Zoo of tree methodologies

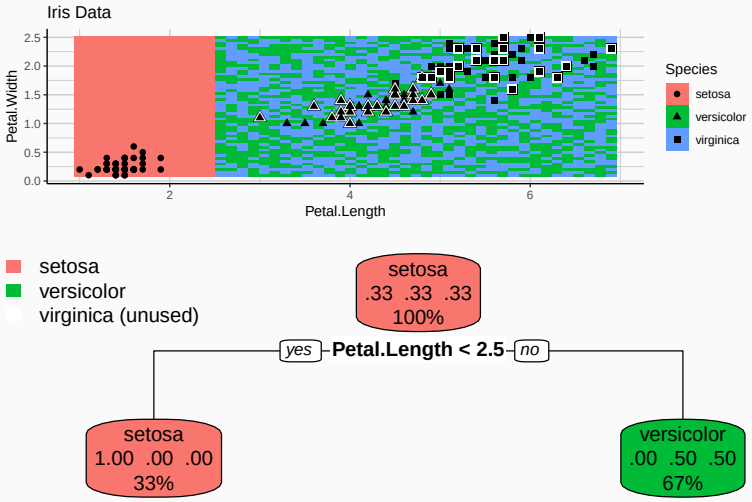
- AID (Sonquist and Morgan, 1964)
- CHAID (Kass, 1980)
- CART (Breiman et al., 1984)
- C4.5 (Quinlan, 1993)
- Unbiased Recursive Partitioning (Hothorn et al., 2006)

Tree Model and Prediction

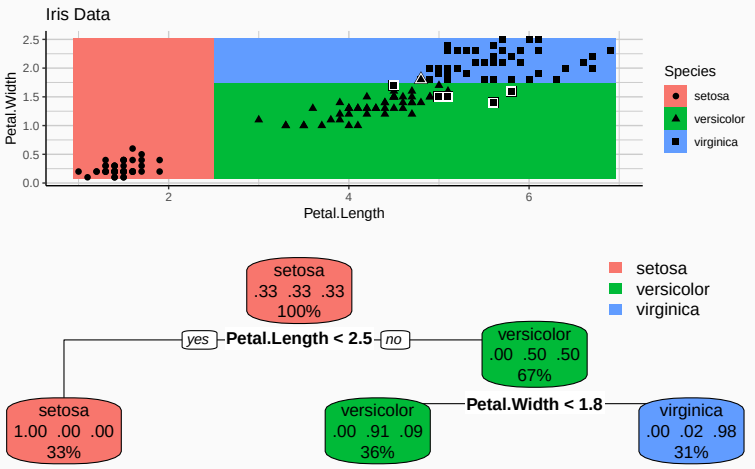
- Classification and Regression Trees, introduced by Breiman
- Binary splits are constructed top-down
- Only constant prediction in each leaf, either a numerical value, a class label or a probability vector.



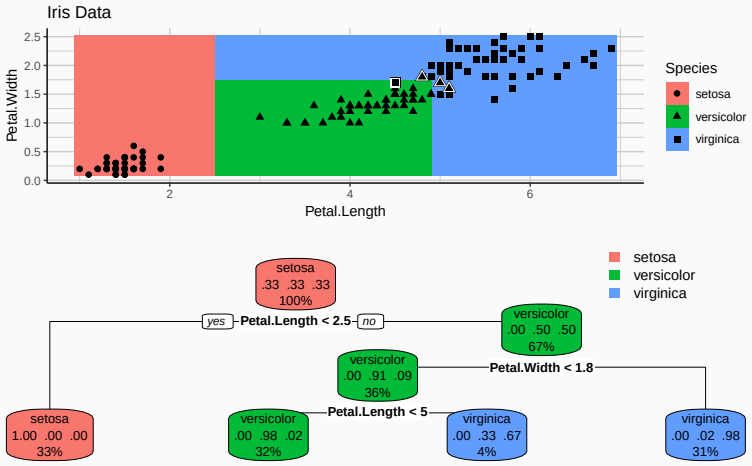
Tree Growing



Tree Growing



Tree Growing

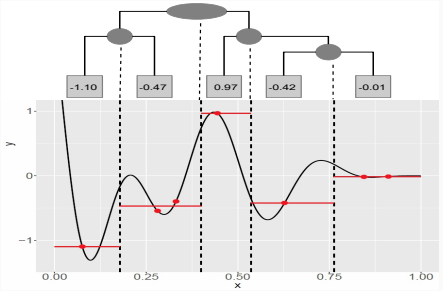


Tree Growing

- In the greedy top-down construction, features and split points are selected by exhaustive search.
- For each node, one iterates over all features, and for each feature over all split points.
- The best feature and split point, which make both created child nodes most pure, measured by a split criterion, are selected.
- The procedure then is applied to the child nodes in a recursive manner.

Example for Regression

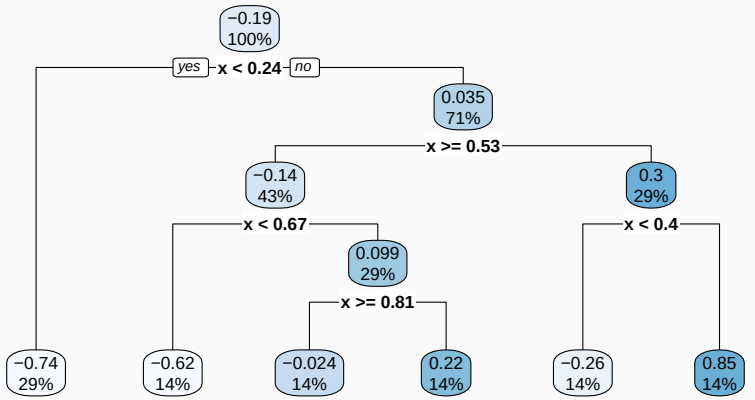
x	y
0.063	-0.794
0.146	-0.691
0.342	-0.258
0.460	0.854
0.602	-0.621
0.744	0.223
0.876	-0.024



Data points (red) were generated from the underlying function (black):

$$\sin(4x - 4) \cdot (2x - 2)^2 \cdot \sin(20x - 4)$$

Example for Regression



Split Criteria

Let $\mathcal{N} \subseteq \mathcal{D}$ be a parent node with two child nodes $\mathcal{N}_1$ and $\mathcal{N}_2$.

Dividing all of the data with respect to the split variable x_j at split point t, leads to the following half-spaces:

$$\mathcal{N}_1(j, t) = \{(x, y) \in \mathcal{N} : x_j \leq t\} \text{ and } \mathcal{N}_2(j, t) = \{(x, y) \in \mathcal{N} : x_j > t\}.$$

Assume we can measure the impurity of the data in node $\mathcal{N}$ (usually the label distribution) with function $I(\mathcal{N})$. This function should return an "average quantity per observation".

Splits are evaluated via impurity reduction:

$$I(\mathcal{N}) - \frac{|\mathcal{N}_1|}{|\mathcal{N}|} I(\mathcal{N}_1) - \frac{|\mathcal{N}_2|}{|\mathcal{N}|} I(\mathcal{N}_2)$$

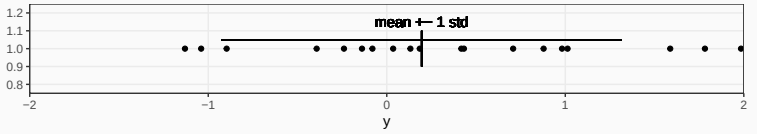
$|\mathcal{N}|$ means number of data points contained in (parent) node $\mathcal{N}$.

Split Criteria - Regression

Mean-squared error / variance

$$I(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{(x,y) \in \mathcal{N}} (y - \bar{y}_{\mathcal{N}})^2 \text{ with } \bar{y}_{\mathcal{N}} = \frac{1}{|\mathcal{N}|} \sum_{(x,y) \in \mathcal{N}} y.$$

Hence, the best prediction in a potential leaf $\mathcal{N}$ is the mean of the contained y-values, i.e. impurity here is variance of y-values.

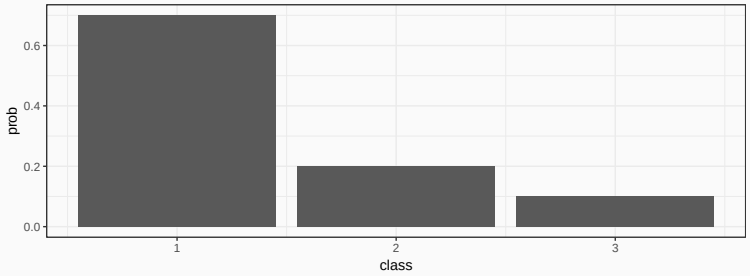


Split Criteria - Classification

Let $\hat{\pi}_k^{\mathcal{N}}$ be the relative frequency of class k in leaf N.

- Leaf predicts the majority label or a probability vector with entries $(\hat{\pi}_1^{\mathcal{N}}, \ldots, \hat{\pi}_g^{\mathcal{N}})$

We now need to measure the "variance" of



Split Criteria - Classification

- Impurity: Misclassification error

$$I(\mathcal{N}) = 1 - \max_k \hat{\pi}_k^{\mathcal{N}}$$

Probability of error in leaf when using the majority label

- Impurity: Gini index:

$$I(\mathcal{N}) = \sum_{k \neq k'} \hat{\pi}_k^{\mathcal{N}} \hat{\pi}_{k'}^{\mathcal{N}} = \sum_{k=1}^{g} \hat{\pi}_k^{\mathcal{N}} (1 - \hat{\pi}_k^{\mathcal{N}})$$

Probability of error in leaf when predicting random label with probabilities $\hat{\pi}_k^{\mathcal{N}}$

- The Gini index is usually preferred, as it is more likely to produce very pure nodes and is less brittle than the misclassification error

Stopping Criteria

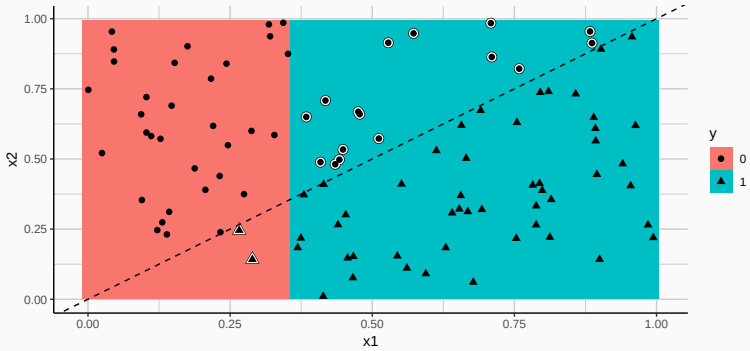
Stopping criteria

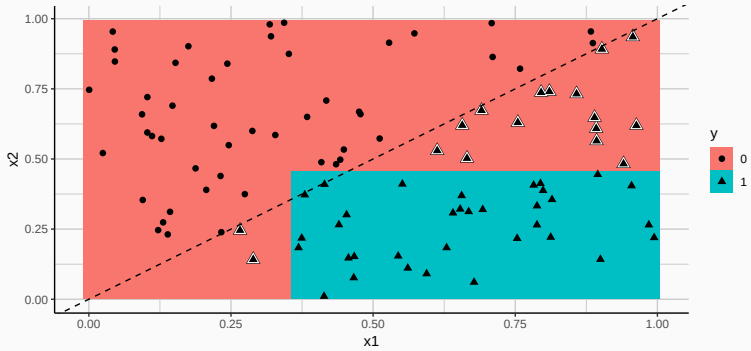
(with control parameter names from package `rpart`)

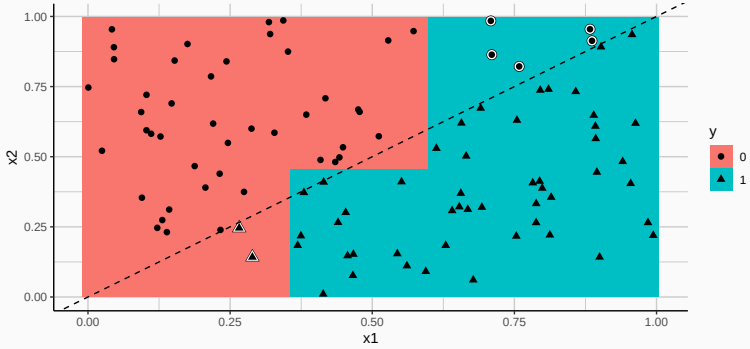
- Min. number of observations per node (`minsplit`)
- Min. number of observations contained in a leaf (`minbucket`)
- Min. increase in goodness of fit (`cp`)
- Max. number of levels for tree (`maxdepth`)

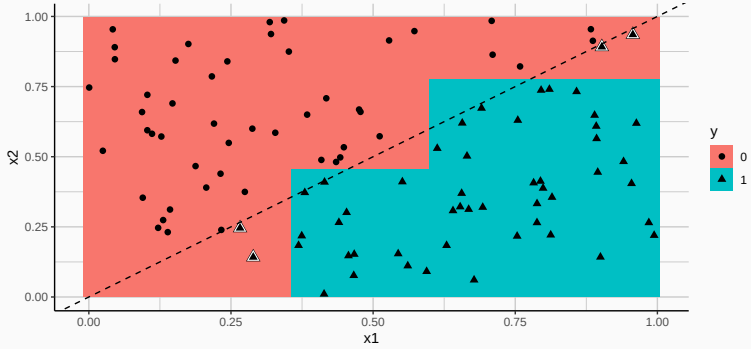
Advantages

- Model is easy to comprehend due to graphical visualization
- Tree structure reflects stepwise decisions
- Interactions between features can be modeled
- Works for non-linear functions as well
- Robust vs. feature outliers or skewed feature distributions
- Built-in feature selection
- Can handle missing values
- Fast implementations exist for large data sizes
- Flexible via custom splits, custom trees for many ML tasks can be built

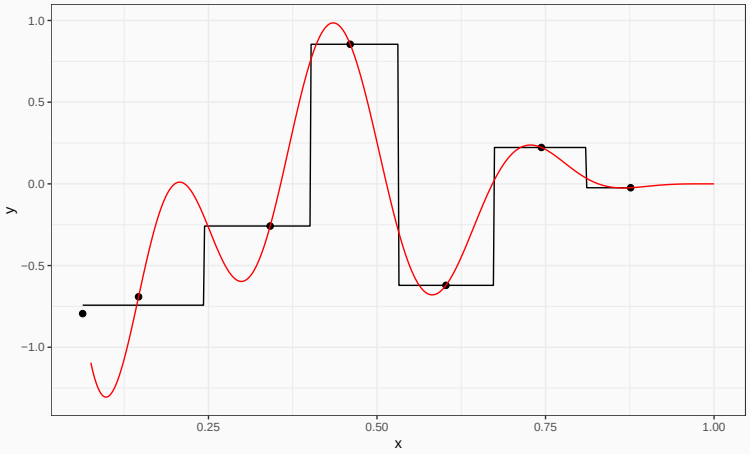

1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
0
1


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
0
1


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
0
1


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
0
1

Disadvantages - Step Function



Prediction function isn't smooth because a step function is fitted.

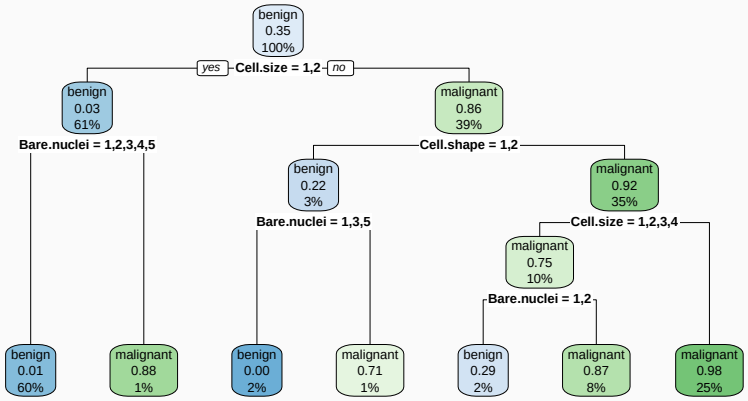
Disadvantages - Instability

High instability of trees will be demonstrated using the Wisconsin Breast Cancer data set. It has 699 observations on 9 features and one target class with values "benign" and "malignant".

Feature name	Explanation
`Cl.thickness`	Clump Thickness
`Cell.size`	Uniformity of Cell Size
`Cell.shape`	Uniformity of Cell Shape
`Marg.adhesion`	Marginal Adhesion
`Epith.c.size`	Single Epithelial Cell Size
`Bare.nuclei`	Bare Nuclei
`Bl.cromatin`	Bland Chromatin
`Normal.nucleoli`	Normal Nucleoli
`Mitoses`	Mitoses

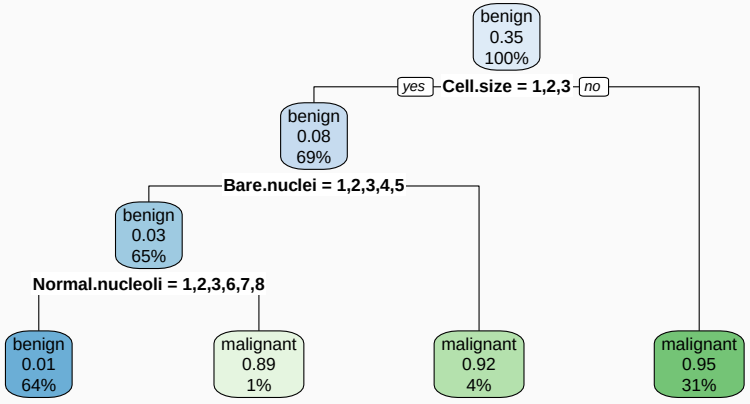
Disadvantages - Instability

Fitted on complete Wisconsin Breast Cancer data



Disadvantages - Instability

Fitted on Wisconsin Breast Cancer data without observation 13

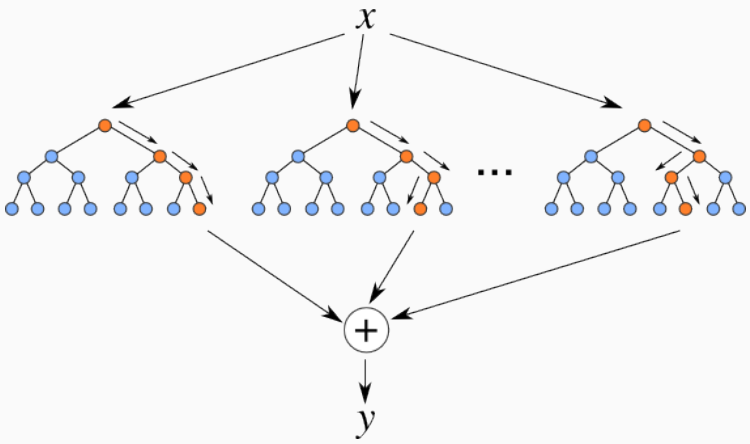


Disadvantages - Summary

- Linear dependencies must be modeled over several splits.
- Prediction function isn't smooth, a step function is fitted.
- High instability (variance) of the trees: Small changes in the data could lead to completely different splits, thus, to completely different trees. Decisions on the upper level influence decisions on lower levels ("mistakes" in upper levels propagate to the lower ones.)
- Really not the best predictor: Combine with bagging (forest) or boosting!

Random Forest

Random Forests



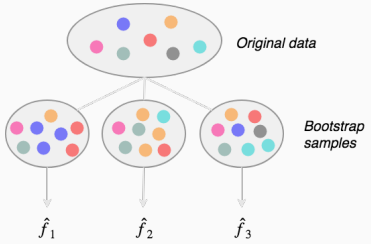
Bagging (I)

- Bagging is based on **B**ootstrap **Agg**regation.
- Ensemble that improves instable / high variance learners
  - Classification and regression trees
  - Neural networks
  - Piecewise variable selection in the regression case, etc.

Bagging (II)

Münchner R Kurse

- Train on B **bootstrap** samples of data D:
  - Draw n observations with replacement
  - Fit the base learner on each of the B bootstrap samples



- **Aggregate** the predictions of the B estimators:
  - Aggregate via averaging or majority voting
  - Posterior probabilities for x in classification can be estimated by calculating class frequencies over the ensemble

Random Forests

Münchner R Kurse

- Modification of Bagging for Trees
- Proposed by Breiman (2001)
- Construction of bootstrapped **decorrelated** trees through randomized splits
- Trees are usually fully expanded, without aggressive early stopping or pruning, to increase variance

Random Forests: Bagging Predictor

Münchner R Kurse

- Variance of the bagging prediction:

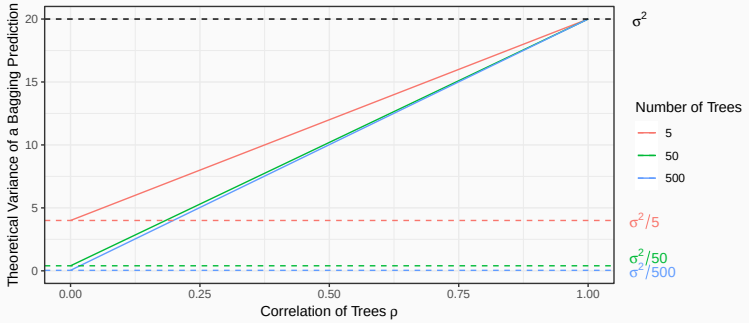
$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 = \left(\rho + (1-\rho)\frac{1}{B}\right)\sigma^2$$

where σ^2 describes the variance of a tree and ρ the positive correlation between trees

  - If trees are highly correlated ($\rho \approx 1$), variance $\to \sigma^2$
  - If trees are uncorrelated ($\rho \approx 0$), variance $\to \frac{\sigma^2}{B}$
  - Variance can be reduced by increasing the number of trees B

Random Forests: Bagging Predictor

Münchner R Kurse



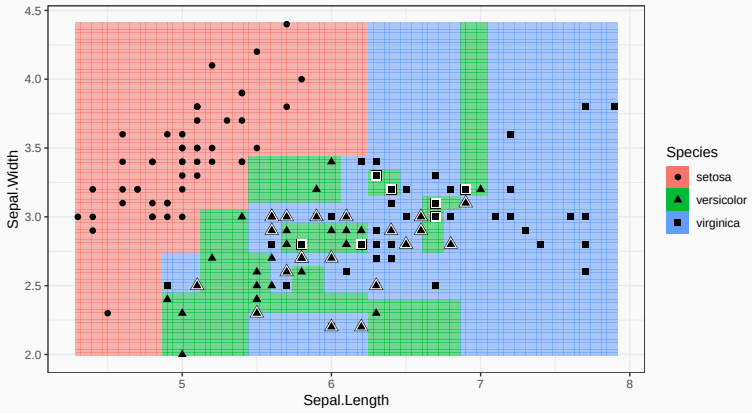
Random Forests: Decorrelation of Trees

Münchner R Kurse

- Draw bootstrap samples
- Instead of all p features, draw mtry $\leq p$ random split candidates. Recommended values:
  - Classification: $\lfloor \sqrt{p} \rfloor$
  - Regression: : $\lfloor p/3 \rfloor$
- Allow trees to slightly overfit by terminating them late

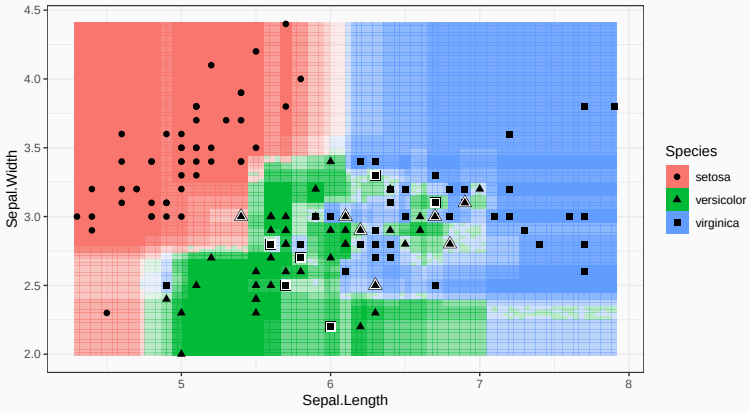
Random Forest With 1 Tree on Iris

Münchner R Kurse



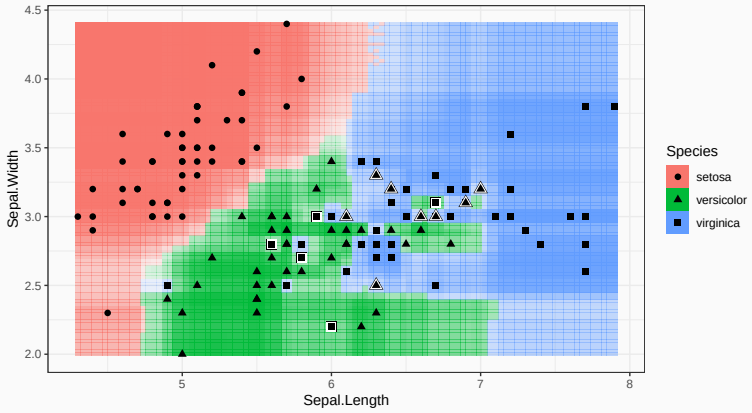
Random Forest With 10 Trees on Iris

Münchner R Kurse

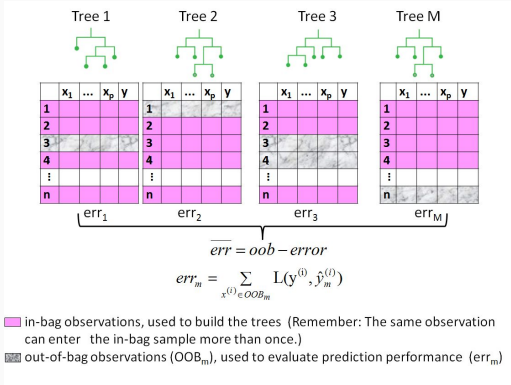


Random Forest With 500 Trees on Iris

Münchner R Kurse



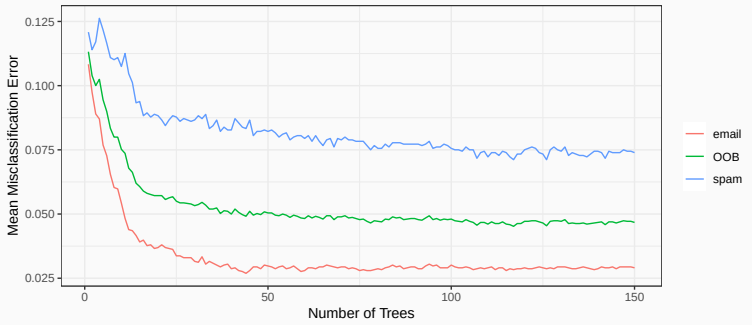
Random Forests: OOB Error



- OOB size: $P(\text{not drawn}) = \left(1 - \frac{1}{n}\right)^n \xrightarrow{n \to \infty} \frac{1}{e} \approx 0.37$
- Predict all x with trees that didnt see it, average error
- Similar to CV, can be used for a quick model selection

Random Forests: OOB Error

Imagine a classification if an email is spam or not. With the Random Forest it is possible to get the mean misclassification errors inherently during the training:



Variable importance

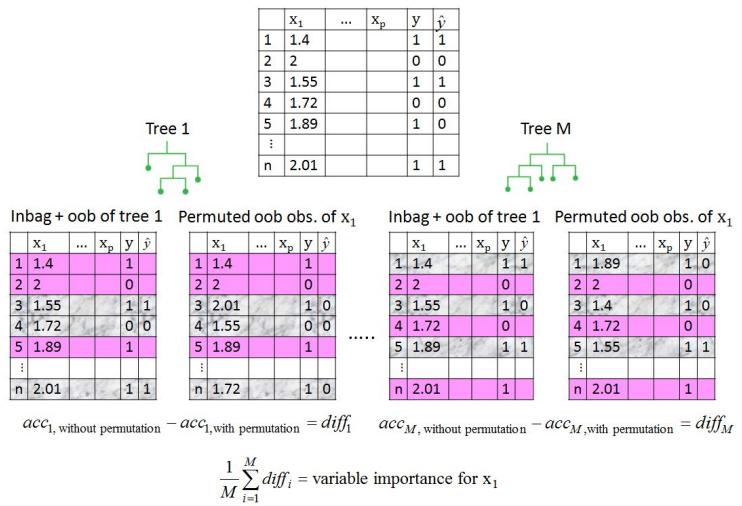
- Single trees are highly interpretable.
- Random Forests as combinations of trees loose this feature.
- Hence, contributions of single covariables to the fit are difficult to evaluate.
- Way out: variable importance measures.

Variable importance

Measure based on permutations of OOB observations:

- While growing tree, pass down OOB observations and record predictive accuracy.
- Permute OOB observations of j-th variable.
- Pass down the permuted OOB observations and evaluate predictive accuracy again.
- The loss of goodness induced by permutation is averaged over all trees and is used as a measure for the importance of the j-th variable.

Variable importance



Advantages

- Bagging is easy to implement
- Can be applied to basically any model
- Easy to parallelize
- Often works well (enough)
- Variable importance: Integrated in RF, and IML package in model agnostic fashion!
- Integrated estimation of OOB error
- Can work on high-dimensional data
- Often not much tuning necessary
- Computationally faster implementation in package `ranger`

Disadvantages

- Often suboptimal for regression
- Does not really optimize loss aggressively in comparison to boosting
- Implementations sometimes memory-hungry
- Prediction can be slow

R package mlr

Motivation: Machine Learning in R

Münchner R Kurse

The **good** news:

- CRAN serves hundreds of packages for machine learning
- Often compliant to the unwritten interface definition:

```
model = fit(target ~ ., data = train.data, ...)
predictions = predict(model, newdata = test.data, ...)
```

The **bad** news:

- Some packages' API is "just different"
- Functionality is always package or model-dependent, even though the procedure might be general
- No meta-information available or buried in docs

Our goal: A domain-specific language for ML concepts!

Motivation: MLR

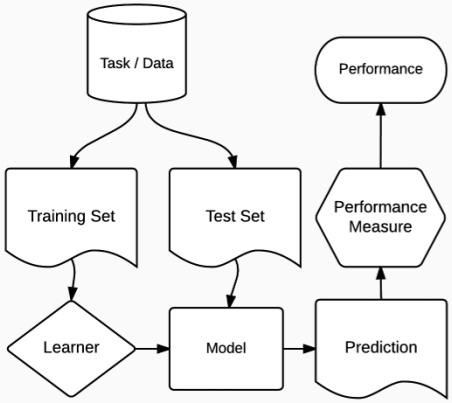
Münchner R Kurse

- Project home page: https://github.com/mlr-org/mlr
  - Cheatsheet for an quick overview
  - Tutorial for mlr documentation with many code examples
  - Ask questions in the GitHub issue tracker
- 8-10 main developers, quite a few contributors, 4 GSOC projects in 2015/16 and one coming in 2017
- About 30K lines of code, 8K lines of unit tests

Motivation: MLR

Münchner R Kurse

- Unified interface for the basic building blocks: tasks, learners, hyperparameters, ...



Basic Features of MLR

Münchner R Kurse

- Tasks and Learners
- Train, Test, Resample
- Performance
- Benchmarking
- Hyperparameter Tuning
- Nested Resampling
- Parallelization

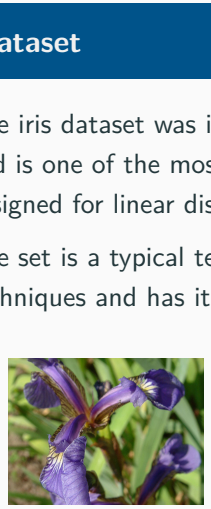
Data Appendix

Prof. Dr. Bernd Bischl

10. September 2018

Used Datasets

Iris Dataset



Setosa

Versicolor

Virginica

The iris dataset was introduced by the statistician Ronald Fisher and is one of the most frequent used datasets. Originally it was designed for linear discriminant analysis.

The set is a typical test case for many statistical classification techniques and has its own **wikipedia page.**

Source: https://en.wikipedia.org/wiki/Iris_flower_data_set

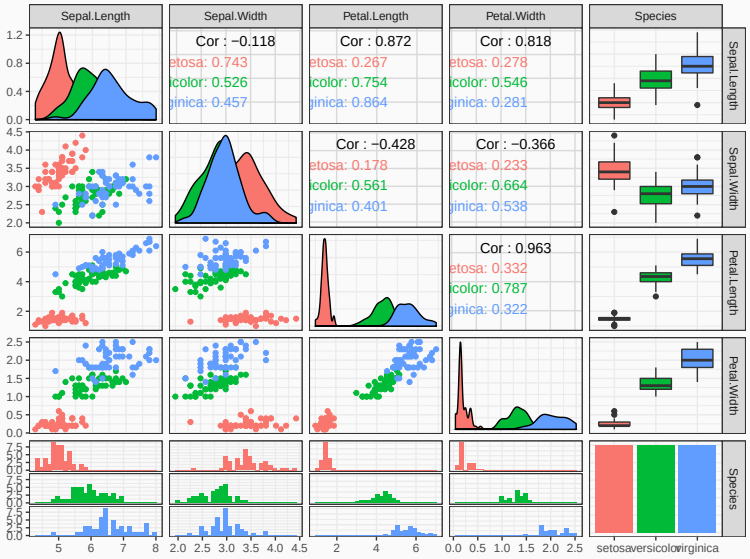
Iris Dataset

- 150 iris flowers (50 setosa, 50 versicolor, 50 virginica), species should be predicted.
- Sepal length / width and petal length / width in [cm].

```
str(iris)

## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1..
```

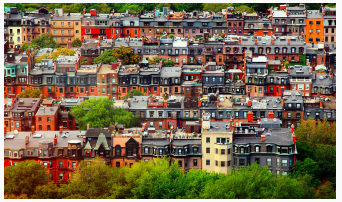
Iris Dataset



Example Data: BostonHousing

A widely used dataset to benchmark algorithms is the Boston housing dataset. The data was originally published 1978 by David Harrison and Daniel Rubinfeld in **Hedonic Housing Prices and the Demand for Clean Air**.

This paper investigates the methodological problems associated with the use of housing market data to measure the willingness to pay for clean air.



Example Data: BostonHousing

- **crim:** per capita crime rate by town
- **zn:** prop. of residential land zoned for lots over 25,000 sq.ft
- **indus:** proportion of non-retail business acres per town
- **chas:** Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- **nox:** nitric oxides concentration (parts per 10 million)
- **rm:** average number of rooms per dwelling
- **age:** proportion of owner-occupied units built prior to 1940
- **dis:** weighted distances to five Boston employment centres
- **rad:** index of accessibility to radial highways
- **tax:** full-value property-tax rate per USD 10,000
- **ptratio:** pupil-teacher ratio by town
- **b:** 1000(B - 0.63)^2 where B is the prop. of blacks by town
- **lstat:** percentage of lower status of the population
- **medv:** median value of owner-occupied homes in USD 1000's

506 obs., 13 features, `medv` numerical target.

Example Data: BostonHousing

```
data("BostonHousing", package = "mlbench")
str(BostonHousing)

## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 ...
##  $ chas   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.52..
##  $ rm     : num  6.58 6.42 7.18 7 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 ...
##  $ rad    : num  1 2 2 3 3 3 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 ...
##  $ b      : num  397 397 393 395 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 ...
```

Spam Dataset

Münchner R Kurse

The spam dataset is one of the datasets used in **The Elements of Statistical Learning** by Trevor Hastie, Robert Tibshirani, and Jerome Friedman. It can be accessed via the `R` package `ElemStatLearn`:

```
library(ElemStatLearn)

##
## Attaching package: 'ElemStatLearn'

## The following object is masked from 'package:plyr':
##
##     ozone

dim(spam)

## [1] 4601   58

table(spam$spam)
```


Spam Dataset

Münchner R Kurse

The feature names of the dataset are named by `A.1` to `A.57` and corresponds to:

- 48 features corresponding to the relative frequency of a specific word in an e-mail
- 6 features that measures the percentage of a sequence of specific characters occurs relative to the total number of characters
- 3 features indicating the average, longest, and sum of uninterrupted sequence of capital letters