

# Chapter 2

## Materials and Methods

### 2.1 Genotype data

Enumerating all genes of a species' genome is one possible level of describing its genotype. It is necessary to group genes of different species according to homology in order enable comparative genomics. Orthologous genes derive from a common ancestor through speciation events and typically share the same biological function. On the other hand, paralogs are evolutionary related through gene duplication, which enables them develop different functions. Orthologous relationships are thus of highest interest for functional inference. Genotypes are represented by clusters of orthologous groups throughout this thesis. COGs are many-to-many relationships among orthologous genes and can be considered taxonomic abstractions of genes. They are delineated from certain patterns in a graph of BBH of all pairwise protein sequence alignments: A minimal COG consists of three BBHs in a triangle. Larger COGs are built by merging those triangles that share common edges. The original COG database [54] featured 720 manually curated COGs based on seven complete genomes and was released in 1997. Due to the greater number of complete genome sequences available today, 722 genomes could be utilized for the current version (2003 COGs 2014 update), which consists of 4632 COGs.

While carefully manually curated databases like the COG database are most valuable tools for biologists, the rapidly growing amount of biological data requires (semi-)automated workflows. The eggNOG database (evolutionary genealogy of genes: Non-supervised Orthologous Groups)

was created in 2007 [55] to close the gap between manually annotated and publicly available genomes. It features exact Smith-Waterman sequence alignments and automated functional annotations. The initial eggNOG release comprised 43 582 general orthologous groups extracted from 373 complete genomes across all domains of life. Roughly a quarter of these NOGs extend existing COGs. The database offers additional NOGs on hierarchically more fine-grained levels, like for example euNOGs for eukaryotes or maNOGs for mammals. EggNOG 1.0 shares its database with the STRING database of protein interactions as of version 7, which is used for the example dataset of the program NetCAR (see Section 2.5).

The updated eggNOG 2.0 was based on 630 complete genomes and yielded 69 221 coarse-grained NOGs (over 200 000 including fine-grained NOGs) [56]. EggNOG 2.0 is a sister database of STRING 8, which served as a resource for the example dataset of the program PICA [18] (see Section 2.6).

EggNOG 4.0 introduced scalable identification of high-quality genomes, quality control procedures, improved functional annotations and further enhancements [7]. The number of high-quality genomes used for NOG delineation (so-called 'core' and 'periphery' genomes) increased to 2031. Some additional 1655 genomes of poorer quality ('adherent' genomes) were also mapped to the orthologous groups. This procedure resulted in 192 421 coarse-grained NOGs. Including the NOGs from all 107 taxonomic levels, there are more than 1.7 million orthologous groups. EggNOG 4.0 data is used for evaluation purposes (Section 3.1.3) as well as for the creation of new phenotype prediction models (Section 3.2) in this thesis. COGs and NOGs will hence collectively be referred to as COGs in this thesis.

Several genomic sequences were used for this purpose that are not covered in eggNOG 4.0. These include genomes recently added to NCBI RefSeq, unpublished genomes obtained from research cooperation partners as well as metagenomic sequence bins. For these, the genotype data was extracted by the following procedure: Nucleotide sequences are processed by PRODIGAL v2.60 for gene calling using the default translation table [57]. The predicted genes are then mapped with the NCBI cognitor software [58] to an in-house generated sequence reference representing all proteins from eggNOG 4.0 COGs.

The current version eggNOG 4.1 features an improved user-interface and several technical improvements. Lacking major changes to the underlying data, it can be interpreted as an

intermediate stage before the release of the next version (presumably eggNOG 5.0), which is expected for late 2015 and will be based on roughly 10 000 genomes.

## 2.2 Phenotype data

The PICA framework provides phenotype labels for its example dataset. They were obtained from the JGI IMG [59] and NCBI Genomes [11] databases as described in [18]. Herein, the same labels are used for reproduction and evaluation purposes of phenotype prediction tools. Ten phenotypes are included in the example dataset. These include aerobic, anaerobic, facultative anaerobic, gram-negative, halophilic, motile, photosynthetic, psychrophilic, endospore-forming and thermophilic traits.

Additional phenotype information was required for novel phenotype models, which were created for methanotrophs, nitrifiers and intracellular microorganisms. It was obtained by manual knowledge extraction from scientific literature by the corresponding cooperation partners for each phenotype (see Section 3.2).

## 2.3 Accuracy Measures

Following the definition from ISO 5725-1, *accuracy* describes a relation between *trueness* and *precision* of a measurement method. Trueness is the distance from the averaged measurements to the *true* value. Precision measures the variability within a set of measurements and thus gives information about the reproducibility of the measurement. Measurement methods of high trueness and high precision are called *accurate*. In binary classification tasks, accuracy is typically defined as ratio of correctly classified cases to the total number of cases. It is calculated as

$$\text{Acc} := \frac{TP + TN}{TP + FP + TN + FN} \quad (2.1)$$

where TP ... true positives, FP ... false positives, TN ... true negatives, FN ... false negatives. In the PICA framework, this measure is called *raw accuracy*. Under certain circumstance,

the linear raw accuracy (Fig. 2.1AD) is not suitable, e.g., when the classes are of heterogeneous sizes or an emphasis on either *selectivity* or *sensitivity* is required. Several other concepts exist, out of which two are presented here, as they are used by netCAR and/or PICA.

*Balanced accuracy* is defined as the mean of selectivity (*true negative rate*) and sensitivity (*true positive rate*, *recall*). It is thus a performance measure that gives penalty, if either of both terms dominates the other (Fig. 2.1BE).

$$\text{BalAcc} := \frac{1}{2} * \left( \frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right) \quad (2.2)$$

The *F1 score* is the harmonic mean of sensitivity and precision. It does not take TN into account, so it is dominated by the *true positive rate* (TPR). In an extreme case of only true negatives, the F1 score would evaluate to zero, even if a classifier works flawlessly (compare Fig. 2.1CF).

$$F1 := \frac{2 * TP}{2 * TP + FP + FN} \quad (2.3)$$

In this thesis, balanced accuracy is the measure of choice, because of ubiquitous dataset imbalance. The F1 score will be used only for the evaluation of netCAR, since it does not support balanced accuracy.

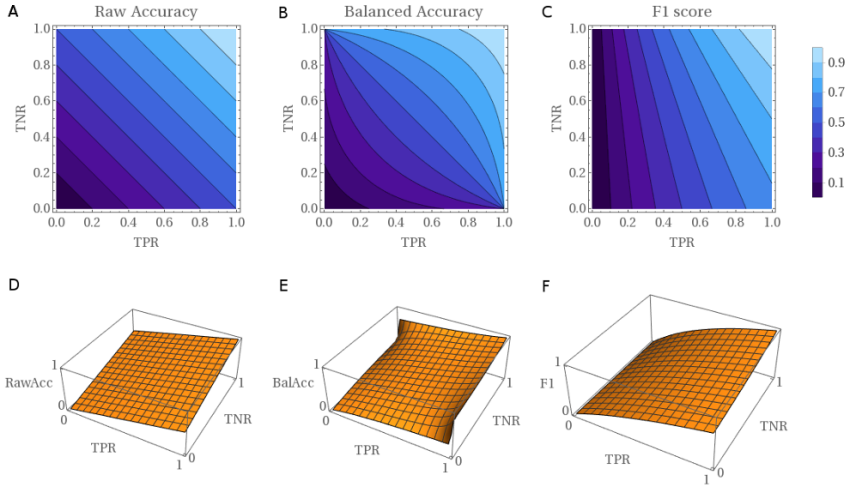


Figure 2.1: Three different accuracy measures and their linear or non-linear dependence on true positive rate (TPR) and true negative rate (TNR) depicted as contour plots and 3D plots.

## 2.4 KRONA

KRONA is a visualization tool that was developed for exploring taxonomy in metagenomic datasets [60]. It is nonetheless also suited for depicting taxonomy of any dataset, utilizing inherent hierarchical information of taxonomic assignments. KRONA visualizations are especially convenient when viewed on a computer, since they are highly interactive and follow *Shneiderman's mantra* (overview first, zoom and filter, details on demand). In static form they still provide a means for comprehensively depicting species of a dataset in their taxonomic context.

## 2.5 NetCAR

NETCAR is a heuristic CAR mining algorithm developed by Tamura and D'haeseleer [17]. It is implemented in a program named netCAR. The notation *NetCAR* will be used in this thesis for both the algorithm and the program. The algorithm was introduced for mining fixed-size

sets of COGs that associate with a phenotype of interest. The program implements two additional algorithms (*Apriori*, *MI-CAR*) for comparative reasons. Enumeration of all possible sets quickly becomes intractable as the number of COGs and set sizes increase, as described in section 1.2.2. It is therefore necessary to sensibly reduce the hypothesis space. Standard algorithms like *Apriori* apply the *downward closure property*, which states that an itemset is frequent, if and only if its subsets are also frequent. *Apriori* thus mines all rules that have the required minimum support. In a sensitive setting of low minimum support, this approach is not efficient enough for large hypothesis spaces. NetCAR uses a COG connectivity graph to counter this problem: Nodes (COGs) are connected, if the corresponding phylogenetic profiles have sufficiently large mutual information (MI)

$$MI(X, Y) = \sum_{x,y} p_{x,y}(x, y) \log \frac{p_{x,y}(x, y)}{p_x(x)p_y(y)} \quad (2.4)$$

where  $p(\cdot, \cdot)$  is a joint probability mass function and  $p(\cdot)$  is a marginal probability mass function. MI is a measure of mutual dependence of two random variables. In the continuous case its unit is the *bit*, if the *logarithmus dualis* is used.

The NetCAR algorithm constructs rules by starting with single COGs (*parents*) that have MI with the phenotype above a certain threshold. Further COGs (*childs*) are considered, whose distance to their parents is less than the requested rule size. Connected subgraphs with at least one parent are subsequently considered as candidate sets. The intersection of their phylogenetic profiles is then used as the set's profile. Mutual information is finally calculated between the COG set phylogenetic profile and the phenotype profile. If it exceeds a certain threshold, a new rule is created with the COG set as antecedent and the phenotype as consequent.

### NetCAR usage

NetCAR is written in the Java programming language and runs on any computer with Java Runtime Environment 1.6 or higher. A procedure with standard settings (rule antecedent size 2, NetCAR algorithm) can be invoked with the following command:

```
java -jar netCAR.jar \
./data/phylogeneticProfile.txt ./data/phenotypeProfile.txt
```

The input file *phylogeneticProfile.txt* contains the items (phylogenetic profile) of the  $i$ -th COG in the  $i$ -th row. The  $j$ -th column corresponds to the  $j$ -th sample (species). Presence and absence are represented as 1 and 0 values, respectively. The text needs to be delimited by comma, space or tab.

*phenotypeProfile.txt* is formatted like the other input file. Its  $j$ -th element indicates binary phenotype classification, where 1 stands for phenotype presence, 0 for absence and -2 indicates a null value used if the phenotype status is unknown.

An output file called *2-Rule.txt* is created at the path `./data`.

More information about NetCAR usage is display by the program itself, when invoked as

```
java -jar netCAR.jar
```

### Changes to NetCAR source code

Three changes have been applied to NetCAR in order to enable the evaluation procedure:

1. Fixed re-usage of the COG connectivity graph:

NetCAR has the option to use a pre-computed COG connectivity graph. This is especially useful, if rules for several different phenotypes are to be mined based on the same phylogenetic profiles, because the construction of this graph is a major contributor to overall computational cost. However, in NetCAR version 1.1, the main class fails to pass the connectivity graph to the NetCAR core algorithm, whose methods in turn do not take the graph as an argument. This was fixed to enable COG connectivity graph re-usage.

2. Fixed violation of transitivity principle by a comparator:

NetCAR reproducibly throws exceptions, if confronted with rules of confidence less than 0.5. This bug is caused by the *RuleComparatorByMI* class, which implements a comparator that does not satisfy the transitivity principle  $A < B \wedge B < C \Rightarrow A < C$ . This was fixed by omitting one conditional.

3. Disabled undocumented output limit for 10,002 rules:

NetCAR has no fixed limit for the number of rules it can mine in one run, however,

there is an undocumented hard-coded limit for 10,002 rules that are written to the output file. While a number of rules greater than this threshold is certainly hardly ever useful, such unexpected behavior can be confusing to the user. Also, in the evaluation, I am interested in the number of rules that are created by the algorithm. This limit was consequently removed.

## 2.6 PICA

PICA is a software framework for phenotype prediction developed by MacDonald and Beiko [18]. It is free open source software under the Creative Commons Attribution-Share Alike 3.0 license and mainly written in the Python 2 programming language. The original release can be obtained from <http://kiwi.cs.dal.ca/Software/PICA>. The framework features plug-ins for different machine learning techniques. In the current release, these are association rule miners, support vector machines and a method based on mutual information. PICA provides scripts for training phenotype models from tagged data (*train.py*), predicting phenotypes of untagged data (*test.py*) and performing cross-validations for parameter selection and model quality estimation (*crossvalidate.py*). They make use of a common PICA application programming interface (API), which allows flexible usage of the framework. PICA operates on the level of presence or absence of genetic features, using COGs by default. The software has not received further improvements by the original developers since the original release in 2010. All changes I applied to PICA have been incorporated into the current release, which can be obtained from the Beiko lab website at <http://kiwi.cs.dal.ca/Software/> or directly from <https://github.com/univieCUBE/PICA>.

### CPAR

PICA implements a heuristic ARM algorithm called CPAR based on the work of Yin and Han [61]. CPAR inherits ideas from traditional rule-based classification algorithms, such as C4.5 or FOIL, as well as from ARM techniques. While the latter may yield higher accuracy, the former are often computationally cheaper. CPAR employs dynamic programming, which avoids redundancy in rule generation. FOIL uses a greedy approach to learn rules that discriminate



between positive and negative examples in a dataset. It iteratively updates a rule and deletes all positive examples that are already comprehended by the current best rule. The algorithm stops, when all positives are considered. The rules are created by repeatedly adding the item  $p$  that brings the highest gain according to

$$\text{Foil gain} := |P^*| \left( \log \frac{|P^*|}{|P^*| + |N^*|} - \log \frac{|P|}{|P| + |N|} \right) \quad (2.5)$$

where  $|P|$  and  $|N|$  are the numbers of positive and negative examples, respectively, covered by the current rule.  $|P^*|$  and  $|N^*|$  indicate the values of the new rule.

CPAR modifies this greedy approach by selecting not only the best item according to Foil gain, but creates several rules at once by also considering items with slightly less gain. The newly created rules are always checked for duplicates in previously built rules. This constitutes a depth-first search in rule generation that avoids generation of a large number of candidate rules. Moreover, this procedure mines rules of variable size by design, since the algorithm only stops when a certain dataset coverage is reached or all items with sufficiently large Foil gain have already been used. This is advantageous compared to NetCAR, which requires the user to pre-define rule size.

## CWMI

An additional approach to phenotype prediction in PICA is based on MI. The standard MI models similar distributions of two variables. In the biological setting genotype features and phenotypes may exhibit similar distributions for two principal reasons:

1. The feature set is a causative agent for the phenotype, thus, whenever the set is observed, the phenotype is to be expected (though not necessarily the other way round).
2. There might be a spurious relationship between the feature set and the phenotype due to common ancestry.

Taxonomy is thus a major confounding factor for phenotype prediction. Conditional mutual information (CMI) can be used to model genotype-phenotype associations under consideration

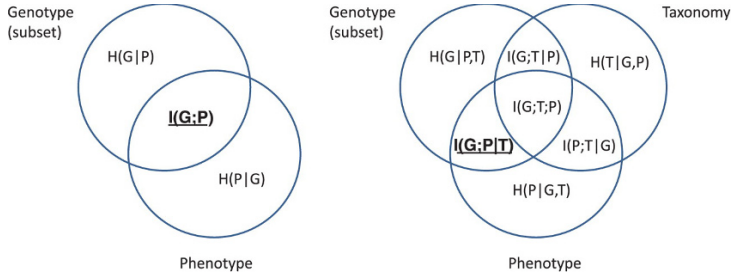


Figure 2.2: Modeling taxonomy as confounding factor in phenotype prediction with CMI. The left diagram visualizes standard MI, as the shared information between a genetic feature set and a phenotype. The right diagram introduces taxonomy as confounding factor. We search for feature sets that maximize  $I(G;P|T)$ , which is the amount of the MI that can not be explained by inheritance from the same ancestors. Figure taken from [18].

of taxonomic effects. It is formally defined as

$$\text{CMI}(X, Y|Z) := \sum_{x,y,z} p(x, y, z) \log \frac{p(x, y, z)p_z(z)}{p_{x,z}(x, z)p_{y,z}(y, z)} \quad (2.6)$$

where general properties match those of the classical MI (Formula 2.4, p. 30). A graphical interpretation of CMI is depicted in Fig. 2.2.

The CWMI score was developed by MacDonald and Beiko to take two aspects into account:

1. Genes of interest share high information that can not be explained by common ancestry.
2. Having identified those, we are interested in the actual MI.

It does so by scaling CMI with the difference of entropy in the phenotype variable and its MI with taxonomy and is defined as

$$\text{CWMI}(X, Y|Z) := \alpha(X, Y, Z)I(X, Y) \quad (2.7)$$

where

$$\alpha(X, Y, Z) := \frac{I(X, Y|Z)}{H(Y|Z)} \quad (2.8)$$

$$H(Y|Z) := \sum_{y,z} p_{y,z}(y, z) \log p(x, y, z) p_{y|z}(y|z). \quad (2.9)$$

## Support vector classification

PICA is shipped with a plug-in for SVC. The current release features purely SVM-based prediction as well as a mixed approach utilizing ARM as a feature-selection step for SVC (called CPAR2SVM). Standard parameters are a linear kernel with soft margin cost parameter  $C=5$ . PICA makes use of the libSVM library [19]. Support vector machines are explained in detail in Section 1.2.3.

## PICA usage

In this section, a typical phenotype workflow is described. It takes changes into account, which I introduced and mainly reduce the number of required command-line arguments by setting sensible standard parameters. The original PICA release necessitates more arguments. All versions give hints at program usage when called with option `python scriptname.py -h` OR `python scriptname.py --help`.

Assume, we have a dataset of five species, which are represented by six genes and labeled for two different phenotypes. The input data would then comprise two files: *genotype.txt* is a file containing the genotype profile, which is essentially a binary matrix, but represented in a sparse tab-separated format: Each line corresponds to one example/species, which is identified by the first column. The following columns hold features/genes, which are present in the species. Absent genes are not stored explicitly, but will be generated by considering set of all genes in the file. Both identifier and features are represented as strings. The file content is the following:

```
Tax1 GeneA GeneB GeneC
Tax2 GeneA GeneC
Tax3 GeneB GeneD
Tax4 GeneD GeneE GeneF
Tax5 GeneA GeneC GeneE
```

The second file, *phenotype.txt*, is constructed similarly. It contains the phenotype profile, which assign YES/NO/NULL labels to each species, corresponding to phenotype presence/-

absence/unknown status. The first column contains the identifiers, which serve as primary keys and thus have to match the identifiers of the other file (arbitrary row order is allowed).

```
Species TraitX TraitY
```

```
Tax1 YES YES
```

```
Tax2 NO YES
```

```
Tax3 YES NO
```

```
Tax4 NULL NO
```

```
Tax5 NO YES
```

Having compiled all necessary input, we can now perform cross-validation in order to estimate the expected model quality. For instance, for phenotype *TraitY* this is invoked by `python crossvalidate.py -s genotype.txt -c phenotype.txt -t TraitY -o cvResult.txt > cvLog.txt`. Accuracy measures are printed to the end of *cvLog.txt*, which also contains additional information, e.g., that dimensionality could be reduced from 6 to 5, because of genes with identical phylogenetic profiles. Per-species statistics can be found in *cvResult.txt*. (Note that this is an extreme case as described in Section 2.3, where the F1-score is low despite perfect classification). The next step is training an SVM model from the whole dataset, which is done by `python train.py -s genotype.txt -c phenotype.txt -t TraitY > trainLog.txt`, which by default creates a model file named *TraitY.rules* alongside three files for conserving the class label map and the feature map (as explained in the next subsection). An optional feature ranking step processes these files and is called by `python svmFeatureRanking.py TraitY.rules > featRank.txt`, which creates a list of most predictive features:

```
Group_ID Score Class
```

```
GeneA 0.999647583602 YES
```

```
GeneC 0.999647583602 YES
```

```
GeneD -0.999647583602 NO
```

```
GeneF -0.000704832796457 NO
```

```
GeneE 6.39679281766e-18 YES
```

```
GeneB 0.0 YES
```

GeneA and GeneC are the best predictors for TraitY, while GeneD is just as informative for phenotype absence. GeneB and GeneE are not predictive for any class.

Finally, we might obtain new data like

```
Tax6 GeneD
```

```
Tax7 GeneA GeneB GeneE
```

stored in the file *novelGenomes.txt* and want to predict *TraitY*. This is achieved by invoking `python test.py -s novelGenomes.txt -m TraitY.rules -t TraitY > prediction.txt`. The following predictions are obtained:

```
Testing on 2 samples
```

```
Organism Predicted
```

```
Tax6 NO
```

```
Tax7 YES
```

### Changes to PICA source code

Several changes were applied to PICA in order to enable broken functionality, add new functionality and improve general code quality. The following is a (non-exclusive) list of changes and bugfixes to PICA:

- Rewrote parts of `libSVMTrainer.py` and `libSVMClassifier.py`, so that the mapping of COGs to SVM features is correct, if COGs are present in test examples, which were not encountered in the training set. Before these changes, the SVM module could not be used for phenotype prediction of novel sequences.

The fix makes use of *pickle* routines to save the dictionary holding the COG-feature mapping created during training alongside the SVM model to tertiary memory. In the prediction phase, this dictionary is read, so that COGs are mapped correctly.

- Hard-linked libSVM 2.90, since the library's API changed in version 3.x.
- Increased robustness of input file handling (fixed crashes on empty lines, tabs at *End Of File*, etc. in genotype and phenotype files).

- Improved error handling (message to the user, if input files do not exist or some arguments were not passed to the program, instead of crashing).
- Changed test.py, so that it does not require a phenotype input file. Unless used in a validation scheme, the phenotype profiles of the genomes to be classified are unknown. The requirement is therefore meaningless in actual phenotype prediction scenarios.

### SVM feature ranking

In addition, I extended the software by implementing a function to extract the most predictive features from linear SVM models based on weights as described by Chang and Lin [62]. The source code of the Python script `python svmFeatureRanking.py`, can be found in the supplement (Listing A.1, p. 97). Dumping and ranking features enables the user to interpret the models from a biological perspective: the COGs with the highest discriminative power for presence or absence of a phenotype are ranked at the top. If specific proteins are already known to be part of the genetic underpinning of a phenotype of interest, the user finding the according COGs ranked highly in the list might gain confidence in the biological relevance of the predictions. Unexpected high-ranking COGs hint at previously unknown protein functions associated with the phenotype. However, these may also derive from taxonomic correlations rather than a shared phenotype, which can be caused by insufficient training data.

COGs with identical scores were treated as one feature by PICA, because they have identical phylogenetic profiles. This must be taken into account, when an SVM model shall be interpreted with respect to causal relationships between features and phenotypes.

## 2.7 GenTraitor

GenTraitor (*Generic Trait Predictor*) is the working title for an integrated phenotype prediction pipeline. The goal is to develop software that automatically performs all necessary steps for phenotype prediction and only requires DNA sequence information (genome, metagenomic bin) as input. The first step of this process is annotation of genotypic features. As of now, this involves gene calling with PRODIGAL v2.60 using the default translation table [57] and

mapping them with the NCBI cognitor software [58] to an in-house generated sequence reference representing all proteins from eggNOG 4.0 COGs. This procedure requires expensive PSI-BLAST [63] computations. Other procedures might be implemented to decrease computational cost. For example, Hidden Markov model (HMM) profiles might be used to search for COGs in the sequence data directly. Alternative concepts of genotypic features, like for instance Pfam protein families [64] could be utilized as well and are currently investigated by a colleague. Next, the pipeline would check the genotype data for genome completeness, e.g. based on marker genes or more sophisticated tools like CheckM [65] for metagenomic data. This information may be used to exclude those models that are known to perform bad for the given completeness value or to give warnings about such issues. The following step is the actual phenotype prediction, which is performed by PICA for all selected models. Finally, the predictions are presented to the user. While this pipeline can be used on demand, it would also be possible to broaden the scope and classify for example all genomes deposited in the Ref-Seq database. Ideally, the results of these predictions would finally be deposited in a publicly available online database. At the moment, the different tasks have not yet been integrated as GenTraitor, but are still executed manually.



<http://www.springer.com/978-3-658-14318-3>

Machine Learning for Microbial Phenotype Prediction

Feldbauer, R.

2016, XIII, 110 p. 29 illus., Softcover

ISBN: 978-3-658-14318-3