



BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

A Robust Approach for Discovering Functional Dependencies using Machine Learning Approaches

von

Philipp Jung

Philipp Jung
Matrikelnummer: 872855
16.03.2019

Gutachter:
Prof. Felix Biessmann
Dr. Zweit Gutachterin

ABSTRACT. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Contents

1	Introduction	1
2	Theory	2
2.1	Relational Database Theory	2
2.1.1	Relation Scheme, Relation and Tuples	2
2.1.2	Keys	3
2.1.3	Definition of a Relational Database	3
2.1.4	Definition of a Functional Dependency	3
2.2	FDs in Application	4
2.2.1	Normalization	4
2.3	Limitations of FDs	6
2.3.1	FDs relaxing on the Extent	7
2.3.2	FDs Relaxing on the Attribute Comparison	8
2.4	Robustness of FDs	9
2.4.1	Measuring Robustness	10
2.5	Machine Learning Classifier Theory	12
2.5.1	Datawig Imputer	13
3	Experiments	16
3.1	FD Imputer	16
3.2	Datawig Imputer	17
3.3	Overfitting the Datawig Imputer	18
3.4	Comparing Datawig Imputer with FD Imputer	18
3.5	Dependency detection	20
3.5.1	Dependency stability	20
3.5.2	Results on various Datasets	21
4	Discussion	22
4.1	Begriffsdiskussion	22
	References	22

1 Introduction

Data-driven methods change the way computer scientists approach algorithmic problems. Rather than designing and implementing complex algorithms themselves, recent advances in machine learning have allowed for learned algorithms. While these learned algorithms come with their own limitations and problems, e.g. lack of explainability, some of them have proven to solve classic algorithmic problems in a more performant fashion.

Kraska et al. showed in their 2018 publication “The case for Learned Index Structures” that different index structures can be replaced by learned ones, greatly improving performance.¹

In the field of data cleaning and data enrichment, HoloClean lead the way for machine-learning approaches in the domain of data cleaning. HoloClean is agnostic of the way the database to be cleaned is structured, making it versatile.

One important concept in relational database theory is the idea of *functional dependencies*. Functional dependencies stem from the early days of relational databases. Historically, they were introduced to formalize schema normalization, where a normalized schema is one where no functional dependency between two non-key columns exists. In the past, functional dependencies found broader interest in data analysis and data cleaning.

¹<https://arxiv.org/abs/1712.01208>

2 Theory

Functional Dependencies (FDs) are a way of expressing “a priori knowledge of restrictions or constraints on permissible sets of data” [Mai83, p. 42] in relational database theory. Having been introduced in the 1970s for schema normalization of relational databases, FDs have proven to be useful in a multitude of domains.

In this section, *functional dependencies* and the theoretical foundation necessary to put them into context are introduced. Building on this foundation, *relaxed functional dependencies* are introduced as a way how *robustness* of FDs has been increased in the past. Lastly, machine learning classifier theory will be discussed, introducing the *Datawig* package.

2.1 Relational Database Theory

In order to give a definition of FDs, they need to be put in context to the domain they stem from: relational database theory. Some basic concepts will be introduced in this section.

2.1.1 Relation Scheme, Relation and Tuples

A *relation scheme*² R is a finite set of *attribute names* $\{A_1, A_2, \dots, A_n\}$, where to each attribute name A_i corresponds a set D_i , called *domain* of A_i , $1 \leq i \leq n$. Let $D = D_1 \cup D_2 \cup \dots \cup D_n$, then a *relation* r on relation scheme R is a finite set of mappings $\{t_1, t_2, \dots, t_p\}$ from R to D :

$$t_i : R \rightarrow D,$$

where we call those mappings *tuples* under the constraint that [Mai83, p.2]

$$t(A_i) \subseteq D_i.$$

In application, attribute names are commonly called *column name* or *column attribute*. One can think of them as labels of data that is stored in the respective column.

²also called *relational schema* in literature[Abe+19, p.21]

2.1.2 Keys

A *key* on a relation r on a relation scheme R is a subset $K = \{B_1, B_2, \dots, B_m\}$ with the property that for any tuple $t_i \in \{t_1, t_2, \dots, t_3\}$ the relation

$$t_i(B_k) = t_j(B_k) \Rightarrow t_i \equiv t_j$$

holds for any single $B_k \in K$. In other words, any K -value of a tuple identifies that tuple uniquely. [Mai83, p. 4]

Having defined both *relation scheme* and *keys*, it is now possible to introduce the more complex concepts of relational databases and functional dependencies.

2.1.3 Definition of a Relational Database

When real-world data used by one or multiple application/s is stored on a machine according to the relational model, it is usually stored in a relational database. According to the definition of a relation scheme R , one can formally introduce databases and database schemes:

We assume that R is composed of two parts, S and K . We call S a *set of attributes* and K a *set of designated keys* and describe this composition by writing $R = (S, K)$. A *relational database scheme* \mathbf{R} over \mathbf{U} can now be defined as a collection of relation schemes $\{R_1, R_2, \dots, R_p\}$, where $R_i = (S_i, K_i)$, $1 \leq i, j \leq p$,

$$\bigcup_{i=1}^p S_i = \mathbf{U}.$$

We demand that $S_i \neq S_j$ if $i \neq j$.

A *relational database* d on a *database scheme* \mathbf{R} is a collection of relations $d = \{r_1, r_2, \dots, r_p\}$ such that for each relation scheme $R = (S, K)$ in \mathbf{R} there is a relation r in d such that r is a relation on S that satisfies every *key* in K . [Mai83, p. 94]

2.1.4 Definition of a Functional Dependency

Consider a relation r on scheme \mathbf{R} with subset $X \subseteq \mathbf{R}$ and a single attribute $A_i \in \mathbf{R}$. A FD $X \rightarrow A$ is said to be *valid* in r , if and only if

$$t_i[X] = t_j[X] \Rightarrow t_i[A] = t_j[A] \quad (1)$$

holds for all all pairs of distinct tuples $t_i, t_j \in r$. [Abe+19, p. 21] We say that X *functionally determines* A [Mai83, p. 43] and name X the *left hand side* (LHS), whilst calling A the *right hand side* (RHS).

left hand side				right hand side
ID	PRENAME	SURNAME	TOWN	ZIP
1	Alice	Smith	Munich	19139
2	Peter	Meyer	Munich	19139
3	Ana	Parker	Munich	19139
4	John	Pick	Berlin	12055
5	John	Pick	Munich	19139

Table 1: Example for a FD.

Considering table 1, one can see that every tuple in the *left hand side* subset of the relation uniquely determines the *right hand side*. For the given example we say that ID, PRENAME, SURNAME, TOWN *functionally determines* ZIP, or $\{ID, PRENAME, SURNAME, TOWN\} \rightarrow ZIP$. [Mai83, p. 43]

If inspected closely, one can discover even more FDs in table 1. For example, $TOWN \rightarrow ZIP$ and $ID \rightarrow ZIP$. Since TOWN and ID are subsets of $\{ID, PRENAME, SURNAME, TOWN\}$, we call $\{ID, PRENAME, SURNAME, TOWN\}$ *non-minimal*. A FD $X \rightarrow A$ is *minimal*, if no subset of X functionally determines A . [Pap+15, p. 2] Thus, $ID \rightarrow ZIP$ and $TOWN \rightarrow ZIP$ are *minimal FDs*.

2.2 FDs in Application

FDs are primarily used in database normalization. [CDP16, p. 1] In 1970, E. F. Codd defined the *relational model* [Cod70] and pioneered many important concepts in database theory. In the following section, selected sections of Codd’s work will be introduced to establish the theoretical background necessary for understanding functional dependencies.

2.2.1 Normalization

When introducing the relational database model in his 1970 article “A relational model of data for large shared data banks”, Edgar F. Codd formalized database normalization

alongside.[Cod70] Describing what will be known to academia as **First normal form** (1NF), Codd states that “problems treated [when normalizing databases] are those of *data independence*”, aiming to protect future users of large databases “from having to know how the data is organized in the machine”. [Cod70, p. 1]

Being designed for as efficient as possible query handling, databases at the time were structured hierarchically or navigationally. While this yielded good performance in times when computing time was very expensive, it came with a heavy cost of complexity: “Teams of programmers were needed to express queries to extract meaningful information. [...] Such databases [...] were absolutely inflexible[y]”. [IBM03]

Update-, insertion- and deletion anomalies can be prevented when normalizing a relational database. [Kle11, p. 75]

First Normal Form A relation scheme R is in *First Normal Form* (1NF), if values in $\text{dom}(A)$ are atomic for every attribute A in R . [Mai83, p. 96] Consider table 2 which represents two relational database schemes. It serves as an example of what is called *atomic* and *compound* data in the Relational Database model. [Cod90, p. 6]

Compound Scheme			Atomic Scheme			
	NAME	ADRESS	PRENAME	SURNAME	TOWN	STREET
1	Alice Smith	Munich, Flurstr.	Alice	Smith	Munich	Flurstr.
2	Peter Smith	Munich, Flurstr.	Peter	Smith	Munich	Flurstr.
3	Ana Parker	Munich, Anastr.	Ana	Parker	Munich	Anastr.
4	John Pick	Berlin, Flurstr.	John	Pick	Berlin	Flurstr.

Table 2: The compound attributes ADRESS and NAME can be split into their atomic components TOWN and STREET as well as PRENAME and SURNAME, respectively.

While the compound scheme’s attributes can be decomposed into several other attributes, whereas an atomic attribute cannot be further split into any meaningful smaller components.

For a database it is said that the database is in 1NF if every relation scheme in the database scheme is in 1NF. 1NF is the very foundation of the Relational Model, where the only type of compound data is the relation.[Cod90, p. 6]

Second Normal Form (2NF) A relation scheme R is said to be in *second normal form* (2NF) in respect to a set of FDs F , if it is in 1NF and every nonprime attribute is fully dependent on every key of R . [Mai83, p. 99] This definition can be extended for databases: A database scheme \mathbf{R} is in second normal form with respect to F if every relation scheme R in \mathbf{R} is in 2NF with respect to F .

Third Normal Form (3NF)

2.3 Limitations of FDs

In the field of dependency detection an extensive body of theory and algorithms for FD detection has been created in the past decades. [Pap+15, p. 1] These mainly consider FDs as defined in formula 1. However, the strict detection of FDs yields results that are solely applicable in a strictly controlled environment.

Real-world datasets that researchers, data scientists or database engineers work with are often *noisy*. This noise has many faces: Entries might be corrupted by missing data, wrongly entered data or incomplete data. Columns might contain data of different kind, where one column could contain the binary representation of an image and another column the contains of a wave-audiofile. In these cases, functionally dependent column-combinations might not be detected as such. This may result in misleading insights when searching for FDs.

Researchers have faced these limitations of FDs in the past in many domains. This lead to the definition of many new kinds of dependencies. While there are RFDs introducing general error measures, others are defined “aiming to solve specific problems” [CDP16, p. 147]. Since all of these *new* kinds of dependencies relax the conditions in equation 1, they are called *relaxed functional dependencies* (RFDs). Caruccio et al. classified and compared 35 different RFDs [CDP16, p. 151].

In order to classify RFDs, Caruccio et al. state that each RFD was “based on its underlying relaxation criteria” [CDP16, p. 149]. They proceed to define two such relaxation criterions. *Attribute comparison* refers to “the type of tuple comparison used on the LHS and RHS”. The relaxation criterion *extent* describes a relaxation of the set of tuples for which a FD is valid.

2.3.1 FDs relaxing on the Extent

The term *extent* is used to describe how a RFD relaxes on the subset of tuples for which the RFD is satisfied.

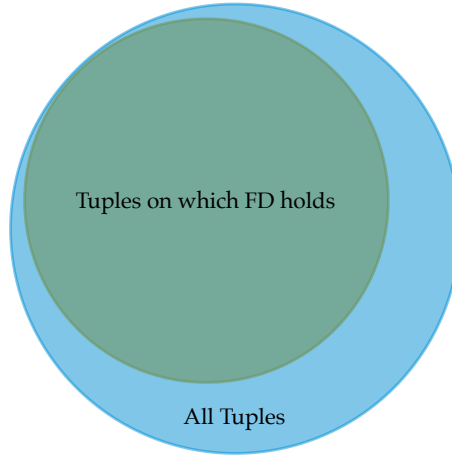


Figure 1: Venn diagram representing the extent relaxation criteria.

RFDs implement this idea in different ways. In the following section, a selection of RFDs relaxing the extent of the FD will be briefly presented.

Approximate Functional Dependency (AFD) AFDs improve the applicability of FDs, by “holding on almost every tuple”[CDP16, p. 151]. An To illustrate this, table 3 shows an example of noisy data. The potential FD $TOWN \rightarrow ZIP$ is not captured by the definition given in equation 1. Due to a type-error, the potential FD is invalidated. To still capture meta-information, a different dependency-measure than given in equation 1 is needed.

Data				
ID	First name	Last name	Town	ZIP
1	Alice	Smith	Munich	19139
2	Peter	Meyer	Muinch	19139
3	Ana	Parker	Munich	19139
4	John	Pick	Berlin	12055

Table 3: Even though column ZIP functionally determines column TOWN (and vice-versa), a FD is not capable of displaying this fact - a typing error invalidates the FD.

Tuples that do not correspond to the canonical FD are measured as fraction of the total of tuples on a relation r as follows:

$$\Psi(X, Y) = \frac{\min(|r_1| \mid r_1 \subseteq r \text{ and } X \rightarrow Y \text{ hold in } r \setminus r_1)}{|r|} \quad (2)$$

Here, function $\Psi(X, Y)$ is called *coverage measure* of a RFD $X \rightarrow Y$. Ψ “quantifies the satisfiability degree of an RFD [...] on r ” [CDP16, p. 150] and is used in the definition of an AFD to be compared to a threshold $\epsilon \in [0, 1]$.

If $\Psi(X, Y)$ is smaller or equal to ϵ , the AFD is said to hold on a relation r . Applied to table 3, the AFD $\text{TOWN} \rightarrow \text{ZIP}$ holds, if $\epsilon \geq 0.25$.

Conditional Functional Dependencies (CFDs) *Conditional Functional Dependencies* employ conditions to define the subset on which a dependency holds. Originally, those conditions exclusively allow the definition of constraints using the equality operator. [CDP16, p. 152] Applied to table 3, a CFD $\text{TOWN} \rightarrow \text{ZIP}$ holds under the condition that a) entries in column $\text{TOWN} = \text{Munich}$ or b) entries in column $\text{TOWN} = \text{Berlin}$.

Other RFDs based on the definition of CFDs include *extended conditional functional dependencies* (ECFDs) by Bravo et al. that allow the disjunction-operator as well as the inequality-operator for defining conditions. [Bra+08]

Also, Chen et al. defined CFD^p s, which introduce $<, \leq, >, \geq$ and \neq for defining conditions. [CFM09]

2.3.2 FDs Relaxing on the Attribute Comparison

Instead of specifying subsets of tuples for which a FD is valid, FDs relaxing on the attribute comparison alter the condition under which a dependency is said to be valid.

Metric Functional Dependencies (MFDs) First introduced by Koudas et al. in 2009, *Metric Functional Dependencies* were defined to adress violations of canonical FDs due to “small variations [...] in data format and interpretation”. [Kou+09, p. 1]

According to equation 1, a FD $\text{TOWN} \rightarrow \text{textsc{Zip}}$ is said to be valid if

$$t[\text{TOWN}] = t'[\text{TOWN}] \Rightarrow t[\text{ZIP}] = t'[\text{ZIP}] \quad (3)$$

holds for all pairs of distinct tuples $t_i, t_j \in r$, where r is a relation on scheme \mathbf{R} . The definition of a MFD replaces this equation by demanding that if and only if for all pairs

of distinct tuples $t_i, t_j \in r$ the metric condition

$$t[\text{TOWN}] = t'[\text{TOWN}] \Rightarrow d(t[\text{ZIP}], t'[\text{ZIP}]) \leq \delta \quad (4)$$

holds, the MFD $\text{TOWN} \rightarrow \text{ZIP}$ is said to be valid on r . [Kou+09, p. 2] Here, δ is called a *tolerance parameter*, indicating how far the result of a metric $d(t[Y], t'[Y])$ can deviate from exact equality.

One such metric d would be the absolute difference between two values, such that $d(t[Y], t'[Y]) = |t[Y] - t'[Y]|$. Another popular metric for measuring the difference between two sequences is the Levenshtein distance, where $d(t[Y], t'[Y]) = \text{lev}_{t,t'}(|t|, |t'|)$ and

$$\text{lev}_{t,t'} = \begin{cases} \max(i, j), & \text{if } \min(i, j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1, j) + 1 \\ \text{lev}_{a,b}(i, j-1) + 1 \\ \text{lev}_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Regarding the example given in table 3, the Levenshtein distance between Munich and Muinch is 2. Thus the MFD $\text{TOWN} \rightarrow \text{ZIP}$ is valid, if $d = \text{lev}$ and $\delta = 2$.

Purity Dependencies (PUDs) *Purity Dependencies* Lorem ipsum

2.4 Robustness of FDs

When Edgar F. Codd introduced the relational model in 1970, the concept of keys and thus the concept of functional dependencies was already present. [Mai83, p. 70] However, in 1972 Codd separated the definition of a FD from keys, persuing the specific goal of normalizing a relational database scheme. In the years to follow, different researchers started using FDs in their works to solve problems other than schema normalization.

Koudas et al., when introducing the MFD, write that the definition of FDs was “not robust³ enough to capture functional relationships on data obtained from merging heterogeneous sources [...]”. [Kou+09, p. 1] However, they do not provide a way of measuring robustness.

Bleifuß et al. [Ble+16, p. 3] define *Correctness* of a set of RFDs *Out* compared to a set of FDs *Gold* on the same relational instance r as follows:

$$\text{Correctness} = \frac{|Out \cap Gold^+|}{|Out|}$$

³Highlighting added by the author.

$Gold^+$ represents the transitive closure of $Gold$, meaning that it includes all non-minimal FDs implied by $Gold$. However, this definition is only meaningful for AFDs, which try to minimize FDs as closely as possible.

As far as the author of this work knows, there is no method for measuring *robustness* of FDs established. Drawing inspiration from Machine Learning theory, we propose the application of cross-validation techniques to functional dependencies in order to measure robustness, as it has been described by Koudas et al. in a meaningful way.

2.4.1 Measuring Robustness

Let $r = \{t_1, t_2, \dots, t_p\}$ be a relation on a relation scheme R . Then $p \in \mathbb{N}$ is the number of tuples in r . We then introduce a split-ratio $s \in (0, 1)$ and split r into two subsets:

$$r_{train} = \{t_1, t_2, \dots, t_{s \cdot p}\}$$

$$r_{test} = \{t_{(s \cdot p)+1}, t_{(s \cdot p)+2}, \dots, t_p\}$$

If $s \cdot p$ is not a natural number, we round accordingly. We call r_{train} the *train set* and r_{test} the *test set*. In table 4 such a splitting is exemplified for $s = 0.5$ and $p = 4$.

A	B	C	D
Blue	Car	Portugal	Lisbon
Yellow	Car	Portugal	Lisbon
Green	Bus	Spain	Madrid
Grey	Car	Portugal	Lisbon

(a) Original relation r .

A	B	C	D
Green	Bus	Portugal	Lisbon
Yellow	Car	Portugal	Lisbon

(b) Train set r_{train} .

A	B	C	D
Yellow	Bus	Spain	Madrid
Blue	Bus	Portugal	Lisbon

(c) Test set r_{test} .

Table 4: The relation r is split into train- and test sets with a split-ratio $s = 0.5$.

In a next step, FDs are detected on the train set using the HyFD algorithm.[PN16] The algorithm we call *FD Imputer* is then used for measuring robustness of each FD detected on the train set.

FD Imputer FD Imputer iterates through each tuple in the test set and searches for a tuple in the train set with the exact same left hand side. If one or more such tuples in the train set are found, FD Imputer randomly selects one of these tuples and saves the right hand side of that tuple. We call this right hand side value *imputation* of the right hand side value on the train set.

This is done for all tuples in the train set. In a last step, FD Imputer compares the imputations with the actual right hand side values in the train set. This allows a classification of the imputations — one can now calculate the F1-measure for each FD.

Consider table 4. The FD $C \rightarrow D$, when used for finding a right hand side value on r_{test} , will correctly return ‘Lisbon’. However, FD $A \rightarrow D$ for example will lead to FD Imputer yielding ‘Madrid’, which is an incorrect imputation.

Implementing FD Imputer, imputations are retrieved by executing a SQL join clause. FD Imputer performs an inner join on all left hand side columns, joining train-set and validation-set. A successive second left join clause concatenates the original validation-set with the column of imputed tuples stemming from the first join.

Robustness as a F1-Score The imputed values retrieved by FD Imputer can be leveraged to create a measure for robustness. Each imputed value is interpreted as a label and classified as in the binary classification case (see figure 2). Precision and recall are then calculated and weighted according to the number of true instances for each label.⁴

Finally, this approach yields a f1-score for each FD which we call *robustness*.

⁴See [sklearn documentation](#) for `sklearn.metrics.f1_score()` with weighted average.

2.5 Machine Learning Classifier Theory

Once a model has been trained and validated, it needs to be tested in order to determine whether or not overfitting occurred during training. This is usually done by measuring the model's performance on a separate dataset not involved in training, the so called test set. Performance is measured according to the type of data and the kind of model involved. To visualize the performance of a classifier, a *confusion matrix* can be created.

Prediction	Ground Truth	
	Positive	Negative
Positive	True Positive	False Positive
Negative	False Negative	True Negative

Figure 2: Illustration of a binary confusion matrix. “Prediction” refers to predicted labels $y_{pred}(x)$ while “Ground Truth” represents the actual labels $y(x)$.

The simplest case of a confusion matrix can be created when measuring the performance of a binary classifier. Figure 2 shows such a binary confusion matrix. Here, “Ground Truth” describes the label $y(x)$ of some data point $x \in X_{test}$, where $y \in \{0, 1\}$. “Prediction” identifies the predicted labels $y_{pred}(x)$ that the model generates after it has been executed on the test-set X_{test} prior unknown.

Whenever $y_{pred}(x) = y(x)$, $x \in X_{test}$ holds, the predicted label can be assigned to be either a *True Positive* (TP) or a *True Negative* (TN). The opposite holds as well, such that a falsely predicted label will be either a *False Negative* (FN) or a *False Positive* (FP).

Using the classification introduced by the binary confusion matrix, all predicted labels y_{pred} are assigned to the four sets TP, TN, FN and FP. Using these four sets, we can introduce measures for classification performance.

Precision is a measure that depicts the proportion of correctly classified positive samples to the total amount of samples classified as positive.[Tha18, p.4] This can be algebraically

expressed as

$$Precision = \frac{|TP|}{|TP| + |FP|} \quad (5)$$

where $|A|$ is the cardinality of a set A . Precision measures how many elements classified as positive are True Positives.

Recall, also called *sensitivity*, represents the share of positive correctly classified samples to the total amount of positive samples.[[Tha18](#), p.3] This can be formalized as

$$Recall = \frac{|TP|}{|TP| + |FN|} \quad (6)$$

Recall measures how many of the positive labelled elements were actually selected.

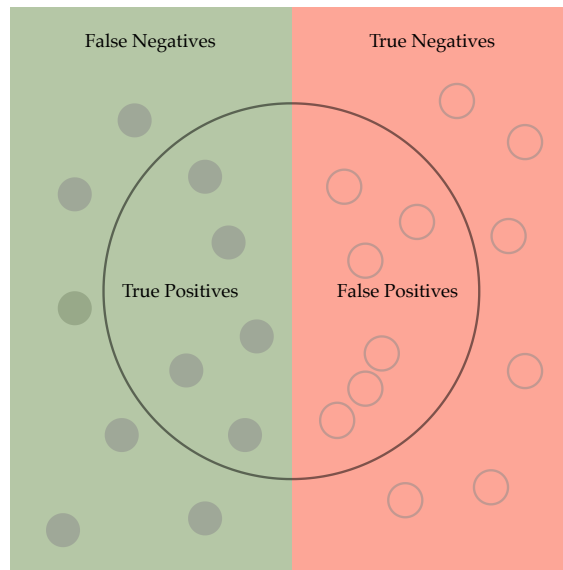


Figure 3: Each predicted label y_x is represented by a circle. Hollow circles stand for negative labels and full circles for positive labels.

The harmonic mean of precision and recall is called *F1-measure*:

$$F1 - measure = \left(\frac{Recall^{-1} + Precision^{-1}}{2} \right)^{-1} \quad (7)$$

2.5.1 Datawig Imputer

Datawig Imputer is an imputation method based on *supervised learning*. In supervised learning, observations are a set of tuples $\{(x_1, y_1), (x_2, y_2), \dots\}$. We call $x_i \in \mathbb{F}$ *feature-vector* in a *feature-space* $\mathbb{F} \subseteq \mathbb{R}^n$ and $y_i \in S$ *output-attributes* or *labels*. [[DHS00](#), p.19]

Datawig Imputer solves a classification problem. A function

$$f : \mathbb{F} \rightarrow S \quad (8)$$

is being approximated, where \mathbb{F} is the feature-space and $S = \{a_1, a_2, \dots, a_M\}$ is a set of attributes.

Datawig Imputer assumes that the column to be imputed, the so-called *label column*, contains data that can be transform to obtain attributes a_1, \dots, a_M . [Bie+18, p.2] All columns on a database table that are *not* the label column are called *feature columns*. These feature columns are what Datawig Imputer will transform to obtain feature-vectors.

The transformation performed to obtain learnable data assign to the string data of each column c a numerical representation x^c . These functions are called *encoders*. Datawig Imputer separates data into two categories: *categorical data* and *sequential data*. Different encoders are chosen to transform *categorical data* and *sequential data*.

Categorical data are data that consist of *categorical variables*. “A categorical variable places a case into one of serveral groups of categories,” define Moore et al. [Moo+11, p. 4] An example-column c_{cat} containing categorical variables can be representend by the following set of colors:

$$c_{cat} = \{\text{blue, red, yellow, blue, blue, red}\} \quad (9)$$

Datawig Imputer creates a histogram of column c and uses the histogram’s index $x^c \in \{1, 2, \dots, M_c\}$ to generate a numerical representation. Thus the encoded column c_{cat}

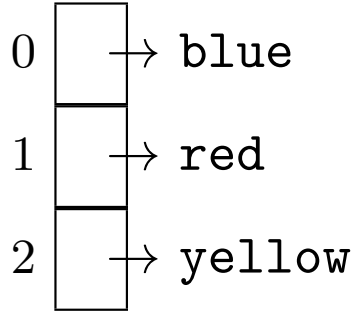


Figure 4: State diagram showing the histogram established by Datawig Imputer to encode example 9

would have the following form:

$$c_{cat}^e = \{0, 1, 2, 0, 0, 1\} \quad (10)$$

Sequential data is data where the sequece in which individual datapoints are stored in contains information. An example for sequential data would be a column containing non-categorical strings, like Usernames:

$$c_{seq} = \{\text{ItalyPaleAle, sisou, primos63}\} \quad (11)$$

The numerical representation $c_{seq}^c \in \{0, 1, 2, \dots, A_c\}^{S_c}$ of the sequential column “is a vector of length S_c , where S_c denotes the length of the sequence or string in column c_{seq} .” [Bie+18, p.2020] This numerical representation is called *n-gram representation*. For further details on the n-gram implementation, consider [Bie+18, p.2020].

Finally, the Datawig-Imputer learns to predict the label distribution from $y \in 1, 2, \dots, D_y$ from the feature-vector x . It therefore models $p(y|x, \Theta)$, the D_y - dimensional probability vector over all possible values in the to-be imputed column in function of feature-vector x and *learned model parameters* Θ . Datawig uses a simple logistic regression type output layer to achieve this:

$$p(y|x, \Theta) = \text{softmax}[\mathbf{W}x + \mathbf{b}] \quad (12)$$

Here, the learned model parameters $\Theta = (\mathbf{W}, \mathbf{z}, \mathbf{b})$ depends on the learned *weights* \mathbf{W} and *biases* \mathbf{b} as well as \mathbf{z} , representing all parameters of the column-specific feature extraction. Parameters Θ are learned by minimizing cross-entropy loss between predicted and observed labels y by computing

$$\Theta = \min_{\Theta} \sum_1^N -\log(p(y|x, \Theta))^{\top} \text{onehot}(y). \quad (13)$$

Here, $\log()$ represents the element-wise logarithm. $\text{onehot}(y) \in 0, 1^{D_y}$ stands for a one-hot encoding of one label y .

Training is done using standard backpropagation and stochastic gradient-descent on mini-batches. The exact network layout is described in the paper “Deep Learning for Missing Value imputation in Tables with Non-Numerical Data” [Bie+18, p.2022] in full detail.

3 Experiments

A number of experiments has been conducted in order to evaluate the capabilities of empirical risk minimization (ERM) techniques for functional dependency discovery. For the experiments, datasets from the UCI Machine Learning Repository were used.[DG17]. All rows containing missing values are dropped due to possible inconsistencies in the algorithms when handling them.

The measure chosen to compare imputation performance on columns containing continuous numeric values is the mean squared error. To compare classification performance, the F1-measure is chosen.

3.1 FD Imputer

FD Imputer is run for every FD found on a train subset of the Adult dataset.[DG17] Figure 5 shows the performance of FD Imputer on columns containing classifiable data. The two top performing FDs have a perfect F1 score of 1 each. An explanation for this can be found when analyzing the content of columns 4 and 5.

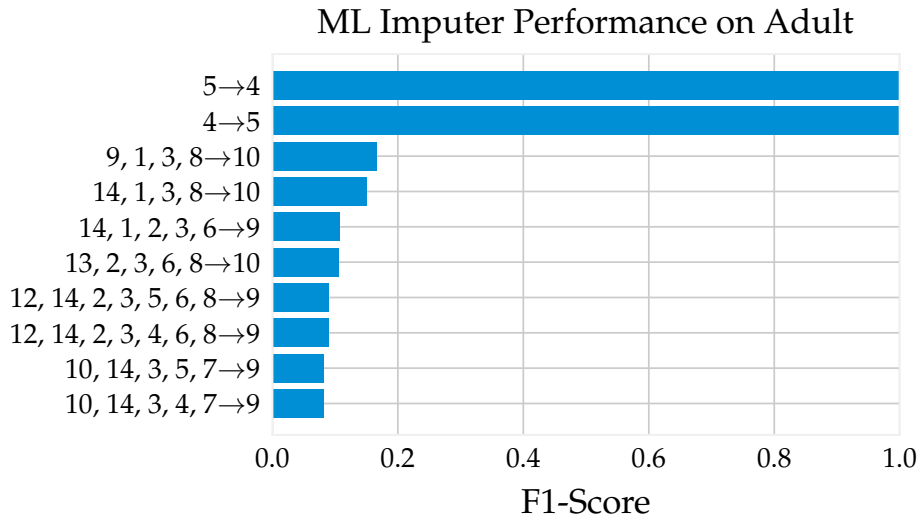


Figure 5: F1 score of the 10 most performant FDs when imputing values on a validation set.

Column 4 contains information about the highest educational level achieved. There are 16 different categories of educational level defined. Each category is assigned an integer in a range from 0 to 15. This integer is the content of column 5. Thus, the relation between column 4 and column 5 can be modeled by a bijective function, projecting the domains of each attribute onto another.

Other FDs lead to F1 scores smaller than 0.2, yielding substantially worse results than

the two top-performing FDs. It can be derived that only the two top-performing FDs hold in a general case. If unseen data is added to the dataset, it can thus safely be assumed that these two FDs still hold.

Dataset	# FDs	# FDs _{train}	Classification Performance			
			F1 _{mean}	F1 _{max}	F1 _{min}	# (F1 = 0)
Adult	93	88	0.0669	1.0000	0.0000	10
Nursery	11	11	0.0000	0.0000	0.0000	10

Table 5: Performance of the FD Imputer on two UCI datasets.

Table 5 provides a summary of how FD Imputer performs on the Adult and Nursery datasets. The column “# FDs” indicates how many FDs were found on the complete dataset. “# FDs_{train}” contains the number of FDs that were detected on the train subset used for imputations. “F1_{mean}”, “F1_{max}” and “F1_{min}” indicate the arithmetic mean, maximal and minimal F1 score achieved on each dataset respectively. The last column named “# (F1 = 0)” provides the number of FDs that scored a F1 score of 0.

The results displayed in table 5 show how strongly the FD Imputer’s performance depends on the dataset considered. On a dataset as normalized as nursery, FD Imputer cannot leverage FDs to impute unknown entries.

3.2 Datawig Imputer

Datawig Imputer is run with the same set of FDs as FD Imputer. The 10 best performing FDs’ F1 scores are displayed in figure 6. The two top scoring FDs are the same for FD Imputer and Datawig Imputer. The third most performant FD is a relation between row ID and nationality. Since there is no apparent dependency between row ID and nationality, it is safe to assume that Datawig Imputer always guesses the most frequent nationality, US-American, for no matter what row ID.

This shows that Datawig Imputer is capable of learning Conditional Dependencies.

Table 6 shows a generally higher performance of Datawig Imputer compared to FD Imputer. There are no FDs for which Datawig Imputer scores 0. This can be explained by to Datawig Imputer’s behavior of always returning some imputation value, even if certainty is very low. This is contrasted by FD Imputer, which frequently returns no imputation value at all.

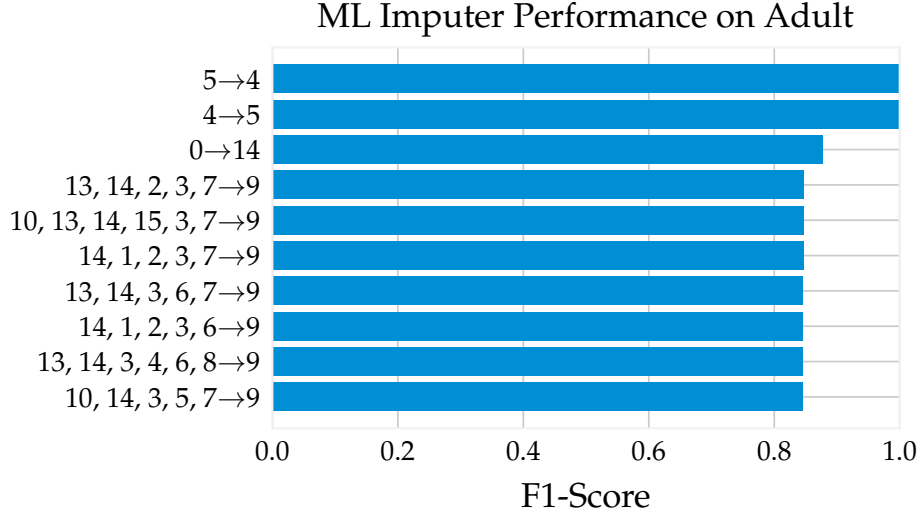


Figure 6: The figure compares the f1-score of the FD Imputer compared to the f1-score of the Datawig Imputer. Each point represents one FD.

Dataset	# FDs	# FDs _{train}	Classification Performance			
			F1 _{mean}	F1 _{max}	F1 _{min}	# (F1 = 0)
adult	93	88	0.7720	1.0000	0.0409	0
nursery	11	11	0.4531	0.9915	0.1138	0

Table 6: Performance of the Datawig Imputer.

3.3 Overfitting the Datawig Imputer

3.4 Comparing Datawig Imputer with FD Imputer

As discussed in the previous section, Datawig Imputer and FD imputer differ fundamentally in the way they function. When comparing the two, metrics need to be computed bearing those differences in mind.

FD Imputer cannot approximate numerical values. Due to the nature of the definition of a FD, Data is always assumed to be classifiable. Meanwhile, Datawig Imputer is able to perform regression, predicting a continuous label for a given input with an uncertainty.

Naturally, this circumstance leads to a far superior performance of Datawig Imputer when imputing continuous labels. FD Imputer usually does not find any values on the train set to impute with and cannot return a meaningful result. Taking the above into account, rows that aren't imputed by FD Imputer are not considered when computing a performance measure on columns containing continuous values.

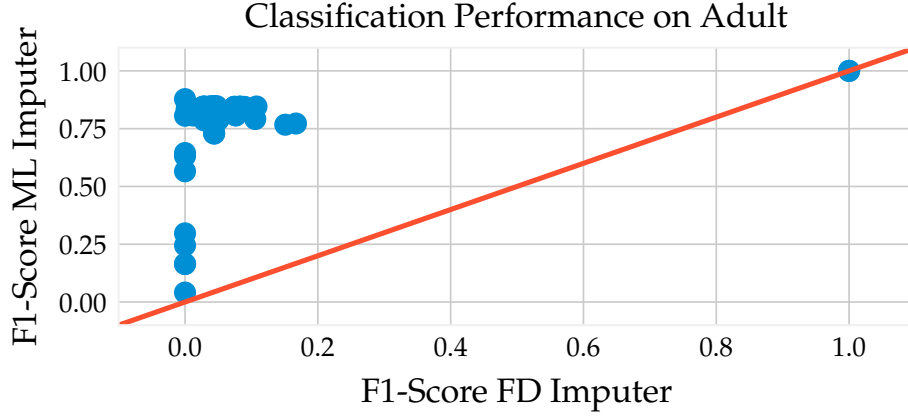


Figure 7: The figure compares the f1-score of the FD Imputer compared to the f1-score of the Datawig Imputer. Each point represents one FD.

Figure 7 compares the F1-scores of both Datawig Imputer and FD Imputer on the Adult dataset. One can observe that for almost all FDs, the Datawig Imputer performs better than the FD Imputer.

FD Imputer performance and Datawig Imputer performance seem to be proportional. If the ML imputers F1-score is lower than 0.7, the FD Imputers F1-score for the same FD is 0. However, for FDs where the Datawig Imputer scores are larger than 0.7, the FD Imputer scores better than 0.0.

There are two FDs for which the FD Imputer performs equally good as the Datawig Imputer. These FDs were identified in the previous section.

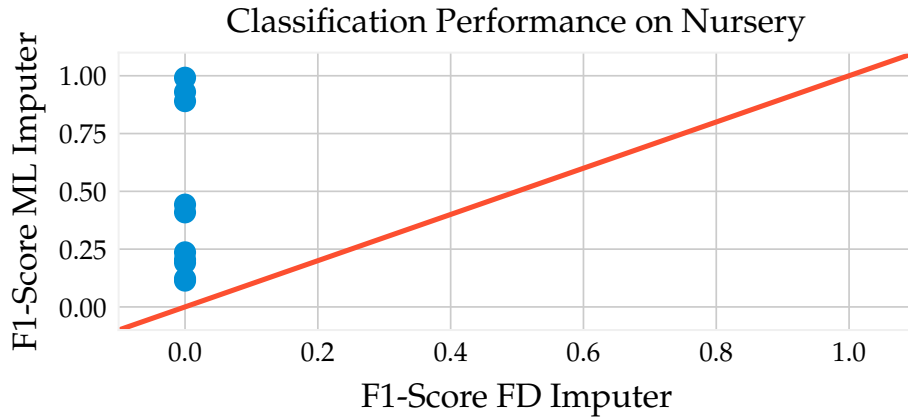


Figure 8: The FD imputers classification performance compared to the performance of the Datawig Imputer on the nursery dataset.

The same experiment was run on the Nursery dataset.[DG17] Results are displayed in figure 8. On this dataset, the ML Imputer performs better for every single FD. This is explainable by the fact that the Nursery dataset is strongly normalized. It contains only 11 FDs, most of them are the key, functionally determining a single column.

3.5 Dependency detection

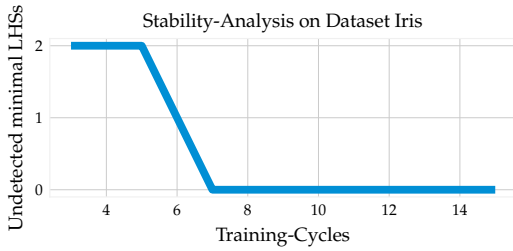
Using the approach described in the theory section to detect a generalized imputation dependency, DepDetector is run on number of datasets. In addition, the stability of discovered minimal dependencies is investigated.

3.5.1 Dependency stability

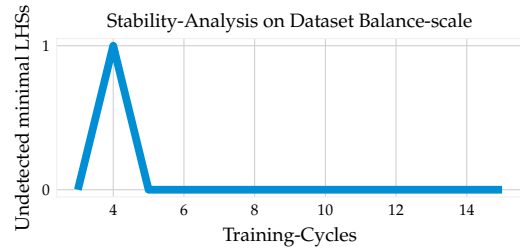
Stability of the found minimal dependency depends on the number of cycles used for training the classifier. DepDetector is run with different numbers of maximal training cycles. Minimal dependencies were searched following the ‘complete’ search strategy.

The number of training cycles τ is increased stepwise from 3 to 15. The number of undetected minimal LHSs is calculated by selecting the result of the run with $\tau = 15$ training cycles as a reference. For every time DepDetector is run to find minimal dependencies, the obtained LHSs are compared to the ones obtained with $\tau = 15$. For each LHS found when $\tau = 15$ that is not contained in the result of a run, the number of undetected minimal LHSs is increased by one.

Figure 9a shows that the Iris dataset contains two minimal LHS combinations and thus two Dependencies. For $\tau \in [7, 15]$ training cycles, DepDetector discovered all minimal dependencies.



(a) Analysis on the Iris dataset reveals that results become continuously more stable for larger τ .



(b) On the dataset Balance-scale, the analysis of the result shows a peak at $\tau = 4$.

Figure 9: Stability analysis of the minimal LHSs found by DepDetector.

An analysis on the dataset Balance-scale shows the potentially non-linear behavior of obtained results in function of τ : Even though the result obtained for $\tau = 3$ is minimal, the result found when $\tau = 4$ is in fact not minimal.

For further analysis, DepDetector models are trained with $\tau = 10$ training cycles, aiming to detect dependencies that are stable.

3.5.2 Results on various Datasets

Dependencies were detected on a number of well-known datasets.

Dataset	Cols	Rows	# FDs	# FDs _{train}	Greedy	Complete
					# F1 _{LHS} > 0.90	# F1 _{LHS} > 0.90
adult	16	32561	93	88	99 (100s)	99
nursery	11	11	99	99	99	99
abalone	10	4177	99	99	99	99
balance-scale	6	625	99	99	99	99
chess	8	28056	99	99	99	99
iris	6	150	99	99	99	99
letter	18	20000	99	99	99	99

Table 7: Result of running dependency detection on selected datasets. Values in brackets are the respective algorithm's runtime in seconds.

4 Discussion

The experiments conducted in the previous section explored the characteristics of FDs. When applied for database scheme normalization, FDs serve a well-defined purpose. However, when it comes to obtaining insights about non-static data, FDs seem to provide little insights about what to expect from new rows that might be added in the future.

FDs do not seem fit for obtaining human-readable information about a (relational) database that is being used in a deployed application. FDs show no resistance to noisy data due to the static nature of their definition. In the field of data profiling, it was proposed that “any dependency can be turned into a rule to check for errors in the data”. [Abe+19, p. 9] This does not seem to be the case in general, but only in a noise-free, static environment.

4.1 Begriffsdiskussion

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

References

- [Abe+19] Ziawasch Abedjan et al. *Data Profiling*. 2019. ISBN: 9781681734477. DOI: <https://doi.org/10.2200/S00878ED1V01Y201810DTM052>.
- [Bie+18] Felix Biessmann et al. ““Deep” Learning for Missing Value Imputation in Tables with Non-Numerical Data”. In: *ACM International Conference on Information and Knowledge Management* 27 (2018), p. 9. DOI: <https://doi.org/10.1145/3269206.3272005>.
- [Ble+16] Tobias Bleifuß et al. “Approximate Discovery of Functional Dependencies for Large Datasets”. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. CIKM ’16. Indianapolis, Indiana, USA: ACM, 2016, pp. 1803–1812. ISBN: 978-1-4503-4073-1. DOI: [10.1145/2983323.2983781](https://doi.org/10.1145/2983323.2983781). URL: <http://doi.acm.org/10.1145/2983323.2983781>.
- [Bra+08] Loreto Bravo et al. “Increasing the Expressivity of Conditional Functional Dependencies without Extra Complexity”. In: *Proceedings - International Conference on Data Engineering* (May 2008), pp. 516–525. DOI: [10.1109/ICDE.2008.4497460](https://doi.org/10.1109/ICDE.2008.4497460).

- [CDP16] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. “Relaxed Functional Dependencies - A Survey of Approaches”. In: *IEEE Transactions on Knowledge and Data Engineering* (2016), pp. 147–165.
- [CFM09] Wenguang Chen, Wenfei Fan, and Shuai Ma. “Analyses and Validation of Conditional Dependencies with Built-in Predicates”. In: (2009), pp. 576–591.
- [Cod70] Edgar F Codd. “A relational model of data for large shared data banks”. In: *Communications of the ACM* 13.6 (1970), pp. 377–387.
- [Cod90] Edgar F. Codd. *The Relational Model for Database Management: Version 2*. Addison-Wesley Publishing Company, 1990. ISBN: 0-201-14192-2. URL: <https://dl.acm.org/citation.cfm?id=77708>.
- [DG17] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.
- [DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience New York, 2000. ISBN: 0471056693.
- [IBM03] Research News IBM. “Former IBM Fellow Edgar (Ted) Codd passed away on April 18”. In: (Apr. 23, 2003). URL: https://web.archive.org/web/20190425094107/https://www.ibm.com/ibm/history/exhibits/builders/builders_codd.html (visited on 06/26/2019).
- [Kle11] Stephan Kleuker. *Grudkurs Datenbankentwicklung*. Vieweg+Teubner Verlag, 2011. ISBN: 978-3-8348-9925-5. URL: <https://link.springer.com/book/10.1007/978-3-8348-9925-5>.
- [Kou+09] N. Koudas et al. “Metric Functional Dependencies”. In: (Mar. 2009), pp. 1275–1278. ISSN: 1063-6382. DOI: [10.1109/ICDE.2009.219](https://doi.org/10.1109/ICDE.2009.219).
- [Mai83] David Maier. *The Theory of Relational Databases*. Computer Science Pr, 1983. ISBN: 0914894420. URL: <http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html>.
- [Moo+11] David S. Moore et al. *The Practice of Statistics for Business and Economics*. W. H. Freeman and Company, 2011.
- [Pap+15] Thorsten Papenbrock et al. “Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms”. In: *Proc. VLDB Endow.* 8.10 (2015), pp. 1082–1093. ISSN: 2150-8097. DOI: [10.14778/2794367.2794377](https://doi.org/10.14778/2794367.2794377). URL: <https://doi.org/10.14778/2794367.2794377>.
- [PN16] Thorsten Papenbrock and Felix Naumann. “A Hybrid Approach to Functional Dependency Discovery”. In: *SIGMOD ’16* (2016), pp. 821–833. DOI: [10.1145/2882903.2915203](https://doi.org/10.1145/2882903.2915203). URL: <http://doi.acm.org/10.1145/2882903.2915203>.
- [Tha18] Alaa Tharwat. “Classification assessment methods”. In: *Applied Computing and Informatics* (2018). ISSN: 2210-8327. DOI: <https://doi.org/10.1016/j.aci.2018.08.003>. URL: <http://www.sciencedirect.com/science/article/pii/S2210832718301546>.