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

IBM's Deep Blue chess-playing computer beat Garry Kasparov in 1997, becoming the first machine to defeat a reigning world chess-champion.[1] IBM researchers implemented alpha-beta search algorithms in parallel, brute-force searching for optimal moves. This approach has been iteratively refined since then, leading to modern chess-engines like Stockfish.

When researchers published the performance of reinforced-learning algorithms in 2018, it became clear that learned algorithms offered superior performanced compared to conventional chess-playing algorithms.[2] This approach, based on empirical risk-minimization, has proven fruitful in domains other than artificial intelligence as well.

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 structures come with their own limitations and problems, e.g. lack of explainability, they 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.[3]

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 versitile.

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.

A long history of academic research improved functional dependency search-algorithms. Most notably, TANE

[1] https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

[2] https://deepmind.com/research/alphago/alphazero-resources/

[3] 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.

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

A *relation scheme*[4] $\boldsymbol{R}$ is a finite set of *attribute names* $\{A_1, A_2, \ldots, A_n\}$, where to each attribute name A_i corresponds a set D_i, called *domain* of A_i, $1 \leq i \leq n$. Let $\boldsymbol{D} = D_1 \cup D_2 \cup \cdots \cup D_n$, then a *relation* r on relation scheme $\boldsymbol{R}$ is a finite set of mappings $\{t_1, t_2, \ldots, t_p\}$ from $\boldsymbol{R}$ to $\boldsymbol{D}$:

$$t_i : \boldsymbol{R} \to \boldsymbol{D},$$

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.

2.1.2 Keys

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

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

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

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

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

2.1.4 Definition of a Functional Dependency

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

$$t_i[X] = t_j[X] \Rightarrow t_i[A] = t_j[A] \tag{1}$$

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, Prenameame, Surname, Town *functionally determines* Zip, or {Id, Prename, Surname, Town} $\to$ Zip. [Mai83, p. 43]

If inspected closely, one can discover even more FDs in table 1. For example, Town → Zip and Id → Zip. Since Town and Id are subsets of {Id, Prename, Surname, Town}, we call {Id, Prename, Surname, Town} *non-minimal*. A FD X → A is *minimal*, if no subset of X functionally determines A. [Pap+15, p. 2] Thus, Id → Zip and Town → 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 undestanding 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 know 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 $dom(A)$ 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]

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 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

	Compound Scheme		Atomic Scheme			
	Name	Adress	Prename	Surname	Town	Street
1	Alice Smith	Munich, Alicestr.	Alice	Smith	Alicestr.	Munich
2	Peter Meyer	Munich, Peterstr.	Peter	Meyer	Munich	Peterstr.
3	Ana Parker	Munich, Anastr.	Ana	Parker	Munich	Anastr.
4	John Pick	Berlin, Johnstr.	John	Pick	Berlin	Johnstr.

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.

dependent on every key of 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)

2.2.2 FD Imputer

The FD Imputer imputes the column of a table of a relational database. Empirical Risk Minimization strategies are used to do this. The table is first split in train-set and validation-set. Then, FDs are detected on the train-set using HyFD.[PN16] Having identified all FDs on the train-set, FD Imputer can impute values of any right-hand side of a particular FD: This is done by executing an SQL join clause. FD imputer performs an inner join on all left-hand side columns, joining train-set and validation-set. A second left join clause concatenates the original validation-set with the column of imputed tuples stemming from the first join.

Algorithm 1: FD Imputer

Result: Validation-subset of a relational database table with an additional column containing imputed tuples
Data: Relational database table

1 Split relational database table into train-set and validation-set
2 Detect FDs in train-set
3 ALTER TABLE train-set DROP COLUMN not in lhs
4 imputed-set = SELECT lhs FROM train-set INNER JOIN validation-set ON lhs
5 imputed-validation-set = SELECT imputed-column FROM imputed-set LEFT JOIN validation-set
6 return imputed-validation-set

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. Howevever, 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, Carucio 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".

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.

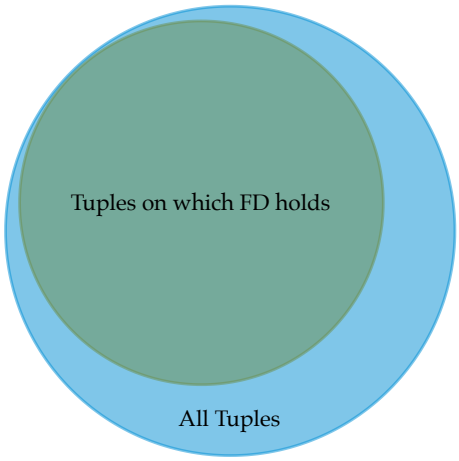

Figure 1: Venn diagram representing the extent relaxation criteria.

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 → 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:

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

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 Town → 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.[**CAR**] Applied to table 3, a CFD Town → Zip holds under the condition that a) entries in column Town = Munich or b) entries in column Town = 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]

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".

2.4 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 occured 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.

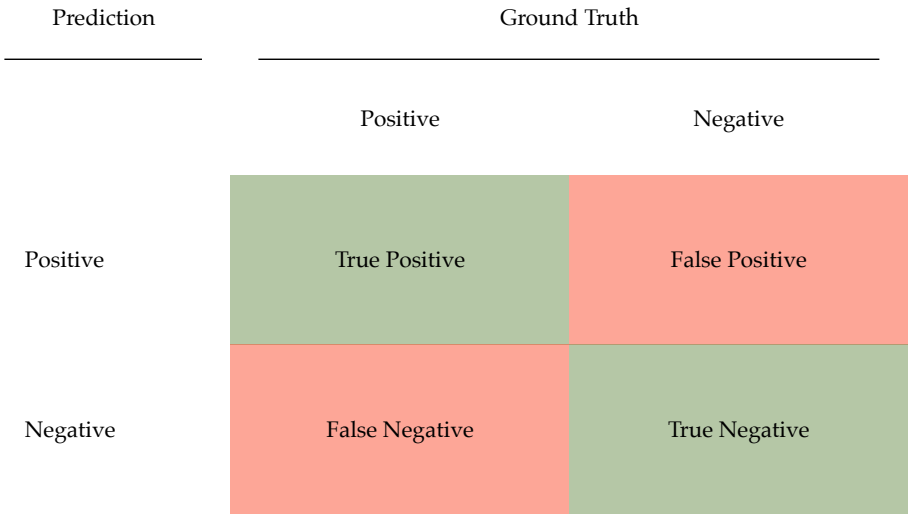

Figure 2: Illustration of a binary confusion matrix. "Prediction" refers to predicted labels $y_{pred}(x)$ while "Ground Truth" represents the actual labels $y(x)$.

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 is 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 algebarically expressed as

$$Precision = \frac{|TP|}{|TP| + |FP|} \tag{3}$$

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

$$Recall = \frac{|TP|}{|TP| + |FN|} \tag{4}$$

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

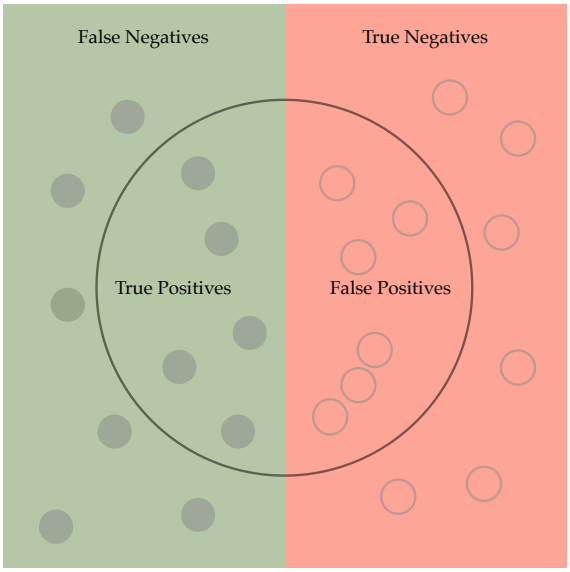

Figure 3: Each predicted label y_x is represented by a circle. Hollow circles stand for negative labels and full circles for positive lables.

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

$$F1 - measure = \left(\frac{Recall^{-1} + Precision^{-1}}{2} \right)^{-1} \tag{5}$$

2.4.1 Datawig Imputer

Datawig Imputer is an imputation method based on *supervised learning*. In supervised learning, observations are a set of tuples $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots\}$. We call $\boldsymbol{x}_i \in \mathbb{F}$ *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

$$f : \mathbb{F} \to S \tag{6}$$

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 comumn*, 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.˜[**"egroup**][p. 4]MOO11. An example-column c_{cat} containing categorical variables can be representend by the following set of colors:

$$c_{cat} = \{\texttt{blue, red, yellow, blue, blue, red}\} \tag{7}$$

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

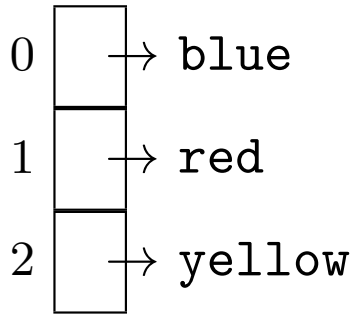

Figure 4: State diagram showing the histogram established by Datawig Imputer to encode example 7

would have the following form:

$$c^c_{cat} = \{0, 1, 2, 0, 0, 1\} \tag{8}$$

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:

$$c_{seq} = \{\texttt{ItalyPaleAle, sisou, primos63}\} \tag{9}$$

The numerical representation $c^c_{seq} \in \{0, 1, 2, \ldots, A_c\}^{S_c}$ 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 $y \in 1, 2, \ldots, D_y$ from the feature-vector $\boldsymbol{x}$. It therefore models $p(y|\boldsymbol{x}, \boldsymbol{\Theta})$, the D_y - dimensional probability vector over all possible values in the to-be imputed comumn in function of feature-vector $\boldsymbol{x}$ and *learned model parameters* $\boldsymbol{\Theta}$. Datawig uses a simple logistic regression type output layer to achieve this:

$$p(y|\boldsymbol{x}, \boldsymbol{\Theta}) = \texttt{softmax}[\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}] \tag{10}$$

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

$$\boldsymbol{\Theta} = \min_{\Theta} \sum_{1}^{N} -\log(p(y|\boldsymbol{x}, \boldsymbol{\Theta}))^{\top} \texttt{onehot}(y). \tag{11}$$

Here, $\log()$ represents the element-wise logarithm. $\texttt{onehot}(y) \in 0, 1^{D_y}$ 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 have been conducted in order to evaluate the capabilities of empirical risk minimization (ERM) techniques for functional dependency discovery. All rows containing missing values are dropped due to possible inconsistencies.

The measure chosen to compare imputation performance on columns containing continuous numeric values is the *mean squared error* (MSE). 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 Adult and Nursery, respectively. 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

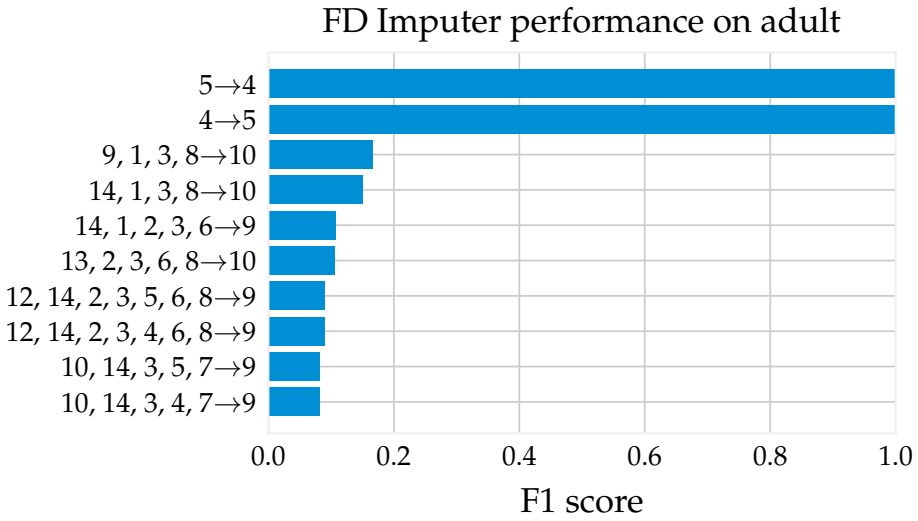

Figure 5: F1 score of the 10 most performant FDs when imputing values on a validation set.

this can be found when analyzing the content of columns 4 and 5. 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 between the domains of each attribute.

Other FDs lead to F1 scores < 0.2, yielding substantially worse results than the top two 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.

Table 4 provides a summary of how FD Imputer performs. The column "# FDs" indicates how many FDs were found on the complete dataset. "# $\text{FDS}_{\text{train}}$" 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

			Classification Performance			
Dataset	# FDs	# FDs_{train}	$F1_{mean}$	$F1_{max}$	$F1_{min}$	# (F1 = 0)
adult	93	88	0.0669	1.0000	0.0000	10
nursery	11	11	0.0000	0.0000	0.0000	10

Table 4: Performance of the FD Imputer.

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 4 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

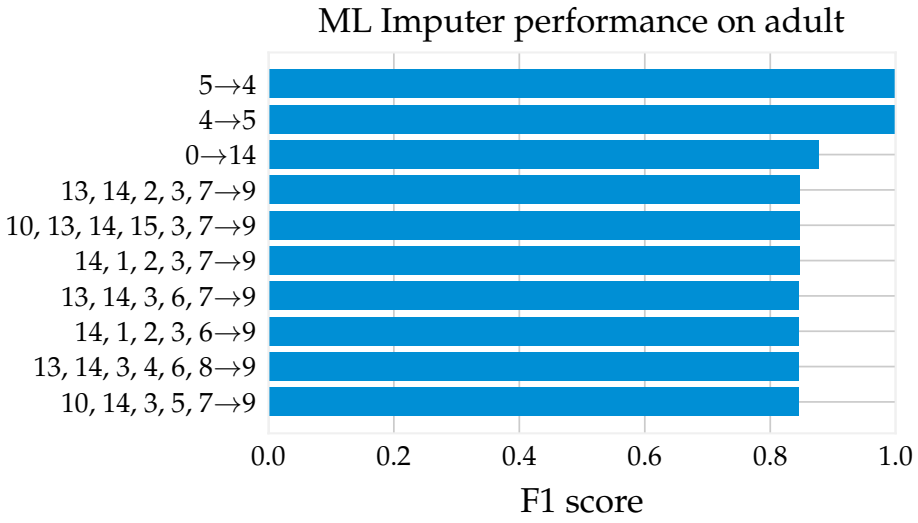

Figure 6: The figure compares the f1-score of the FD Imputer compared to the f1-score of the Datawig Imputer. Each point represents one FD.

Imputer and Datawig Imputer. The third most performant FD is a relation between row ID and nationality. Since there is no appearent 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 5 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

			Classification Performance			
Dataset	# FDs	# FDs_{train}	$F1_{mean}$	$F1_{max}$	$F1_{min}$	# (F1 = 0)
adult	93	88	0.7720	1.0000	0.0409	0
nursery	11	11	0.4531	0.9915	0.1138	0

Table 5: Performance of the Datawig Imputer.

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.

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 doesn't 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.

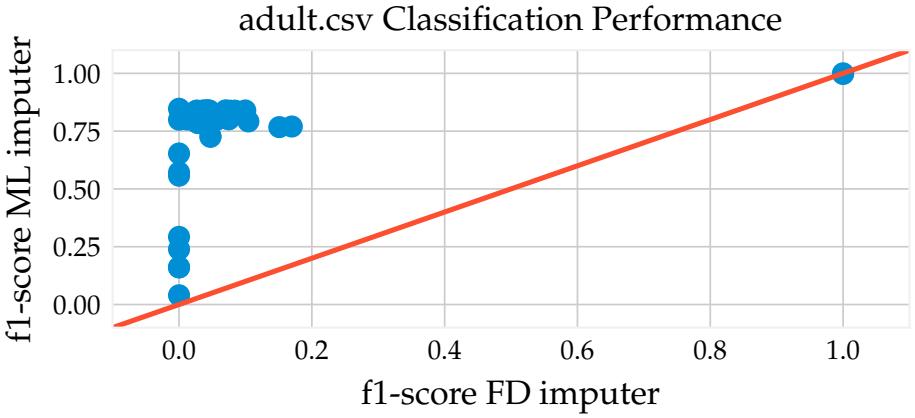

Figure 7: The figure compares the f1-score of the FD Imputer compared to the f1-score of the Datawig Imputer. Each point represents one FD.

Figure 7 compares the F1-scores of both Datawig Imputer and FD Imputer on the Adult dataset. One can observe that for most FDs, the Datawig Imputer performs better than the FD Imputer. FD Imputer performance and Datawig Imputer performance seem to be proportional. If the ML imputer's F1-score is lower than 0.7, the FD Imputer's F1-score for the same FD is 0. However, for FD's where the Datawig Imputer scores are larger than 0.7, the FD Imputer scores better than 0.0. Interestingly, there are two FDs for which the FD Imputer performs equally good or better than the Datawig Imputer.

The same comparison as

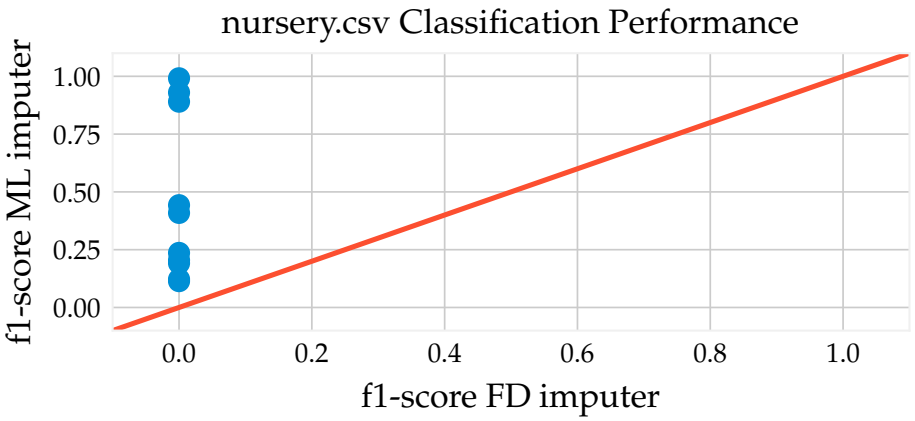

Figure 8: Some ohter caption.

3.5 Dependency detection

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

					Greedy	Complete
Dataset	Cols	Rows	# FDs	# FDs_{train}	# $F1_{LHS} > 0.90$	# $F1_{LHS} > 0.90$
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 6: 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.

[Bra+08] Loreto Bravo et al. "Increasing the Expressivity of Conditional Functional Dependencies without Extra Complexity". In: May 2008, pp. 516–525. ISBN: 978-1-4244-1836-7. DOI: 10.1109/ICDE.2008.4497460.

[CDP16] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. "Relaxed Functional Dependencies - A Survey of Aproaches". In: *IEEE Transactions on Knowledge and Data Engineering* 28.1 (2016), pp. 147–165. ISSN: 2150-8097.

[CFM09] Wenguang Chen, Wenfei Fan, and Shuai Ma. "Analyses and Validation of Conditional Dependencies with Built-in Predicates". In: *Database and Expert Systems Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 576–591. ISBN: 978-3-642-03573-9.

[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.

[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.

[Mai83] David Maier. *The Theory of Relational Databases*. Computer Science Pr, 1983. ISBN: 0914894420. URL: http://web.cecs.pdx.edu/~maier/TheoryBook/TRD.html.

[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. 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. 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.