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* **R** over **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* **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 **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 $X \rightarrow A$ is said to be *valid* in r, if and only if

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

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	FIRST NAME	LAST NAME	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, FIRST NAME, LAST NAME, TOWN *functionally determines* ZIP, or {ID, FIRST NAME, LAST NAME, 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, FIRST NAME, LAST NAME, TOWN}, we call {ID, FIRST NAME, LAST NAME, 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*.

3 FDs in Application

FDs are primarily used in database normalization,[CDP16, p. 1] but also find application in the field of data profiling, where "any dependency can be turned into a rule to check for errors in the data".[Abe+19, p. 9]

3.1 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]

3.1.1 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]

	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.

3.1.2 Second Normal Form (2NF)

A relation scheme R is said to be in *Secon 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 of R.[Mai83, p. 99]

3.1.3 Third Normal Form (3NF)

3.2 Approximate Functional Dependencies

In the field of data profiling an extensive body of theory and algorithms for FD detection has been created in the past decades.[Pap+15] 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 faced by data-scientists or database engineers are often *noisy*. Entries might be corrupted by missing data, wrongly entered entries or incomplete datasets. Inconsistencies are to be expected. Thus, functionally dependent column-combinations might not be detected as such. This may result in misleading insights when searching for FDs.

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.

Approximate FDs (AFDs), sometimes called *Relaxed FDs*, improve the applicability of FDs, "in that they relax one or more constraints of the canonical FDs"[CDP16, p. 1]. While there are AFDs introducing general error measures, others are defined "aiming to solve specific problems"[CDP16, p. 1].

The error measure for this is not trivial at all. While F1-measures can be established for non-categorical cases, comparing results for different data-types tricky.

	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.

3.3 FD Imputer

Algorithm 1: An imputer operating on Functional Dependencies

Result: Imputed column of a relational database
Data: Relational database

```
Split relational database in test-set and train-set
Detect FDs in train-set
for row in test-set do
    Find row in train-set with equal LHS combination
    if matching LHS combination found then
        impute with RHS from train-set
    end
    if No matching LHS combination found in train-set then
        impute with NaN
    end
end
```

4 Execution

A number of experiments have been conducted in order to evaluate the capabilities of empirical risk minimization (ERM) techniques for functional dependency discovery.

4.1 FD Imputer

The FD Imputer

4.2 ML Imputer

4.2.1 Overfitting the ML Imputer

4.3 Comparing ML Imputer with FD Imputer

When comparing the ML Imputer to the FD Imputer, it is necessary to explain the scope of such an comparison. Due to the definition of a FD, the FD Imputer cannot approximate numerical values. Meanwhile, the ML Imputer is able to do so due to the approximative nature of a classifier. To compare classification performance,

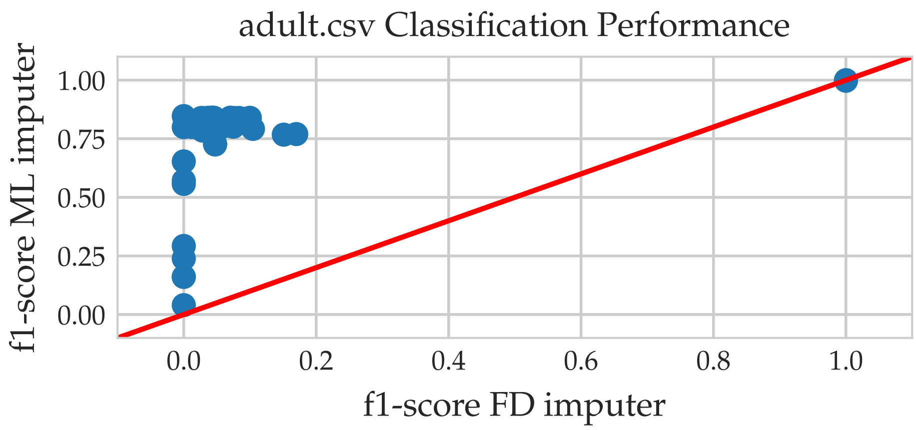

Figure 1: The figure compares the f1-score of the FD Imputer compared to the f1-score of the ML Imputer. Each point represents one left-hand side.

4.4 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.

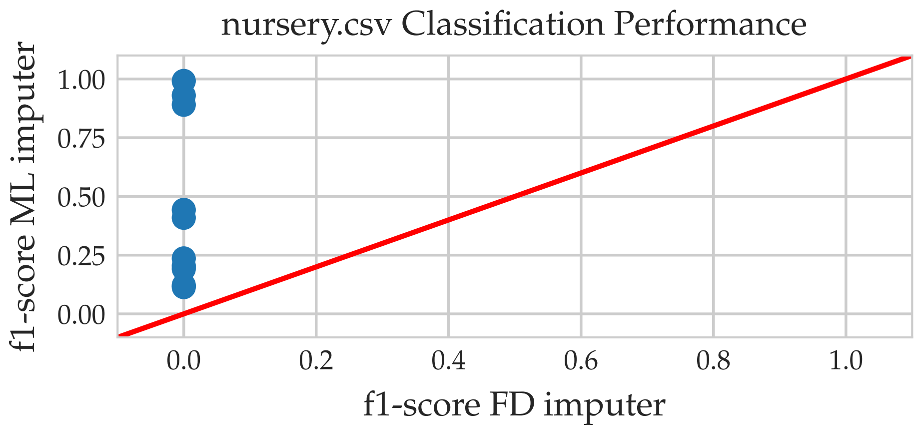

Figure 2: Some ohter caption.

5 Discussion

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.

5.1 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.

[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.

[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.

[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.