Carleton
UNIVERSITY

Combining Declarative Approaches to Data Cleaning with Machine Learning for Entity Resolution: Some New Trends in Data Science

Leopoldo Bertossi[1]

Carleton University → RelationalAI Inc.

Ottawa, Canada

[1]Millenium Institute for Foundational Research on Data (IMFD, Chile)

Prelude on Data Quality

Recent Approaches to Data Quality

- More than a third of data science related tasks involve data assessment and cleaning
- Foundations are less understood and developed than other areas of data management
- Declarative, generic and parameterizable approaches to data cleaning are needed

  Usually *ad hoc*, application dependent and vertical solutions

  Not generalizable or adaptable
- Things are starting to change ...
- Use of contexts for data quality assessment and cleaning

  Data quality is context dependent

  Use of formal and computable ontologies in specification of contexts

- Recently, different forms of data quality constraints have been proposed and investigated
- They provide generic languages for expressing quality concerns

  Suitable for specifying adaptive and generic data quality assessment and data cleaning techniques
- Integration of data cleaning (techniques) with ML and logic-based data processing systems

- Data quality has many dimensions: consistency, completeness, accuracy, redundancy, freshness, ...

  All of them create in the end a problem of uncertainty in data

For a Gist: Characterizing Consistent Data wrt ICs

- Consistency has to do with satisfying semantic constraints, usually in the form of integrity constraints (ICs)
- ICs have been around for a long time ...

  They are used to capture the application semantics in the data model and the database

  They have been studied in general and have wide application in data management

  Much fundamental/technical research has been developed

  Methodologies for dealing with ICs are quite general and have broad applicability

- However, in many situations databases may be inconsistent wrt. a given set of ICs

  Getting rid of violations is sometimes possible, but sometimes impossible or too complex or undesirable

Example: Instance D violates the functional dependency $Name \rightarrow Salary$

Employee	*Name*	*Salary*
	page	5K
	page	8K
	smith	3K
	stowe	7K

- Why not accepting inconsistency, live with it, and make the best we can out of our DB?

- Database repairing and consistent query answering (CQA) are newer contributions in this direction (more coming)

  And more generally, a contribution to a newer approach to data quality problems

- What are the consistent data in an inconsistent database?

  What are the consistent answers to a query posed to an inconsistent database?

- (Arenas,Bertossi,Chomicki; PODS99) provided a precise definition

  Intuitively, the consistent data in an inconsistent database D are invariant under all minimal ways of restoring D's consistency

  Consistent data persists across all the minimally repaired versions of the original instance: the repairs of D

Example: FD: $Name \rightarrow Salary$

$Employee$	$Name$	$Salary$
	$page$	5K
	$page$	8K
	$smith$	3K
	$stowe$	7K

Two possible (minimal) repairs if only deletions/insertions of whole tuples are allowed: D_1, resp. D_2

$Employee$	$Name$	$Salary$
	$page$	5K
	$smith$	3K
	$stowe$	7K

$Employee$	$Name$	$Salary$
	$page$	8K
	$smith$	3K
	$stowe$	7K

$(stowe, 7K)$ persists in all repairs: it is consistent information

$(page, 8K)$ does not (it participates in the violation of FD)

- A consistent answer to a query Q from D can be obtained as a usual answer to Q from every possible repair of D wrt IC

- $\mathcal{Q}_1 : Employee(x, y)?$

  Consistent answers: $(smith, 3K), (stowe, 7K)$

- $\mathcal{Q}_2 : \exists y Employee(x, y)?$

  Consistent answers: $(page), (smith), (stowe)$

- CQA may be different from classical data cleaning!

- However, CQA is relevant for data quality; an increasing need in business intelligence and data science

  It also provides concepts and techniques for data cleaning

- What about a declarative approach to repairs and CQA?

- We can use answer-set programs (ASPa) to specify database repairs

- ASP is a well-established logic programming paradigm

  ASPs used to specify and solve hard combinatorial and optimization problems

  And reason about and query problems with multiple-world semantics (e.g. DB repairs)

Example: Schema $Emp(N, S)$ and FD $N \to S$

$$\neg\exists x z_1 z_2 (Emp(x, z_1) \wedge Emp(x, z_2) \wedge z_1 \neq z_2)$$

Repair program contains the rules: (with global tuple ids, t_i)

$$Emp'(x, z_1, \mathsf{d}) \vee Emp'(x, z_2, \mathsf{d}) \leftarrow Emp(x, z_1), Emp(x, z_2), z_1 \neq z_2$$
$$Emp'(x, z, \mathsf{s}) \leftarrow Emp(x, z),\ not\ Emp'(x, z, \mathsf{d})$$

d, s: annotation constants for "tuple deleted" and "tuple stays in repair", resp.

- A (stable) model M of the program determines a repair D' of D:

  $D' := \{R(\bar{c}) \mid R'(t, \bar{c}, \mathsf{s}) \in M\}$ (and every repair obtained in this way)

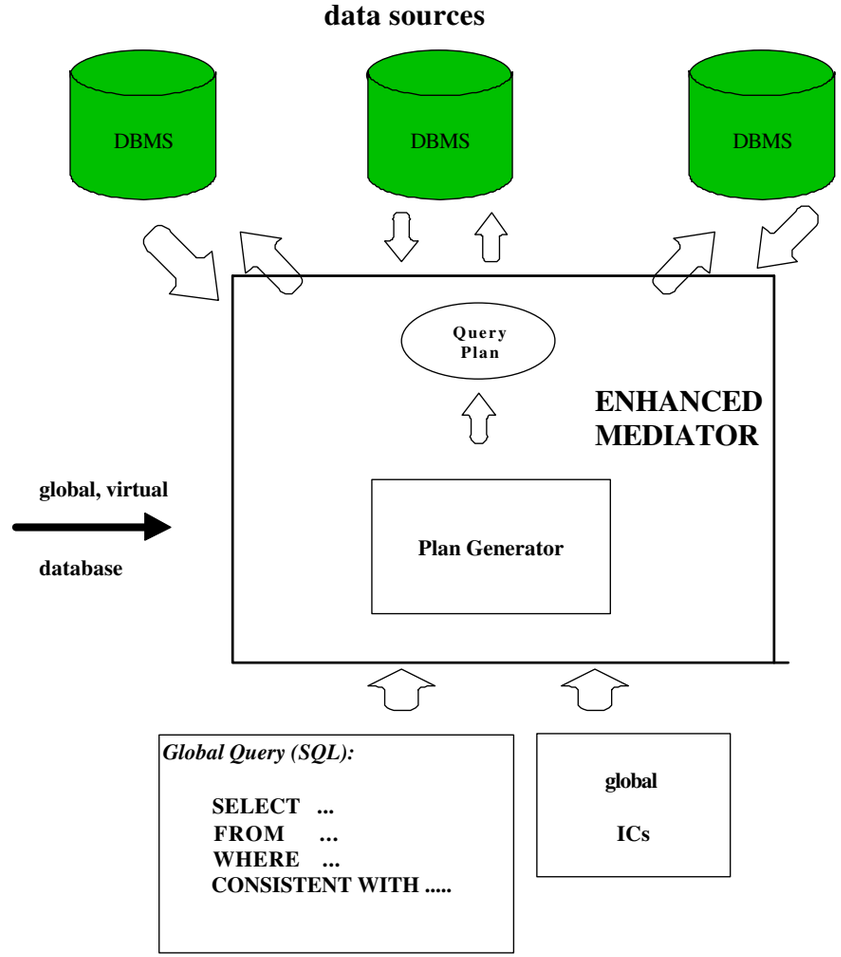
- Natural application scenario:

  Virtual data integration

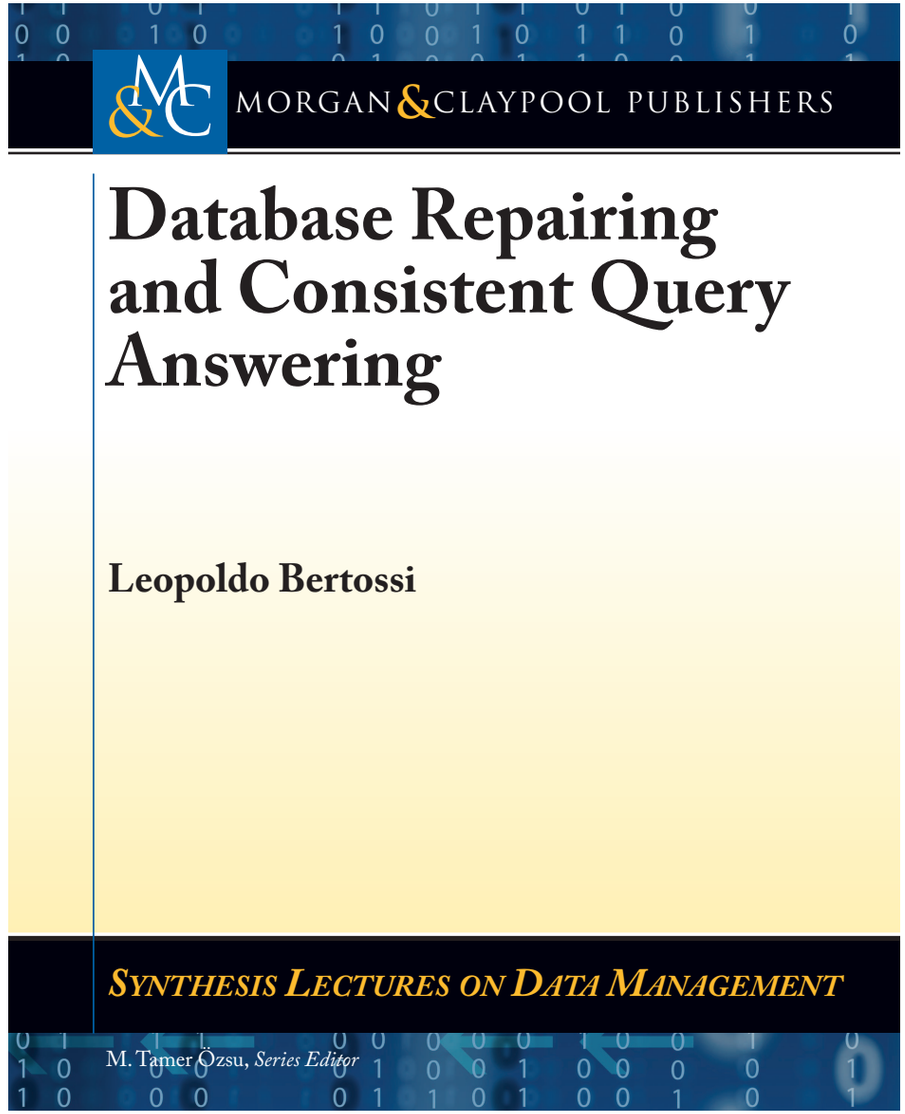
  No way to enforce global ICs on the sources

  Inconsistencies have to be solved on-the-fly, at query- answering time

  [Bravo & Bertossi; IJCAI 2003]



- Newer classes of dependencies have been introduced with data quality/cleaning in mind!


MORGAN & CLAYPOOL PUBLISHERS
Database Repairing and Consistent Query Answering
Leopoldo Bertossi
SYNTHESIS LECTURES ON DATA MANAGEMENT
M. Tamer Özsu, Series Editor

Matching Dependencies (MDs)

- MDs are related to Entity Resolution (ER)
- ER is a classical, common and difficult problem in data cleaning, and ML

  ER is about discovering and merging records that represent the same entity in the application domain

  Detecting and getting rid of duplicates!
- Many *ad hoc* mechanisms have been proposed
- ER is fundamental for data analysis and decision making in BI
- Particularly crucial in data integration
- MDs express and generalize ER concerns

  They are declarative rules with a clear logic-based semantics that help characterized and identify duplicates; and specify and enforce their merging

- MDs specify attribute values that have to be made equal under certain conditions of similarity for other attribute values

Example: Schema $R_1(X, Y), R_2(X, Y)$

$$\forall X_1 X_2 Y_1 Y_2 (R_1[X_1] \approx R_2[X_2] \implies R_1[Y_1] \doteq R_2[Y_2])$$

"When the values for attributes X_1 in R_1 and X_2 in R_2 in two tuples are similar, then the values in those two tuples for attribute Y_1 in R_1 and Y_2 in R_2 must be made equal" (R_1 and R_2 can be same predicate)

$\approx$: Domain-dependent, attribute-level similarity relation

[W. Fan et al.; PODS 2008, VLDB 2009]

- Although declarative, MDs have a procedural feel and a dynamic semantics

An enforcement-based and model-theoretic semantics introduced; and algorithmic/complexity issues investigated

[Bertossi, Kolahi and Lakshmanan; ICDT'11, TOCS 2013]

- An MD is satisfied by a pair of databases (D, D'):

  D satisfies the antecedent, and D', the consequent, where the matching (merging) is realized

  A local, one-step satisfaction ...

- We may need several steps until reaching an instance where all the intended mergings are realized

  Dirty instance: $D \Rightarrow D_1 \Rightarrow D_2 \Rightarrow \ldots\ldots \Rightarrow D'$

  $\uparrow$

  stable, clean instance!

  (there could be several of these)

Example: "similar name and phone number ⇒ identical address"

People	*name*	*phone*	*address*
	John Doe	(613)123 4567	Main St., Ottawa
	J. Doe	123 4567	25 Main St.

⇓

People′	*name*	*phone*	*address*
	John Doe	(613)123 4567	25 Main St., Ottawa
	J. Doe	123 4567	25 Main St., Ottawa

$People(N_1, Ph_1, Ad_1) \wedge People(N_2, Ph_2, Ad_2) \wedge N_1 \approx N_2 \wedge Ph_1 \approx Ph_2 \;\Rightarrow Ad_1 \doteq Ad_2$

A dynamic semantics with a domain-dependent matching function $\mathsf{m}_{address}$

$$\mathsf{m}_{address}(\underline{MainSt., Ottawa}\ ,\ \underline{25MainSt.}) := \underline{25MainSt., Ottawa}$$

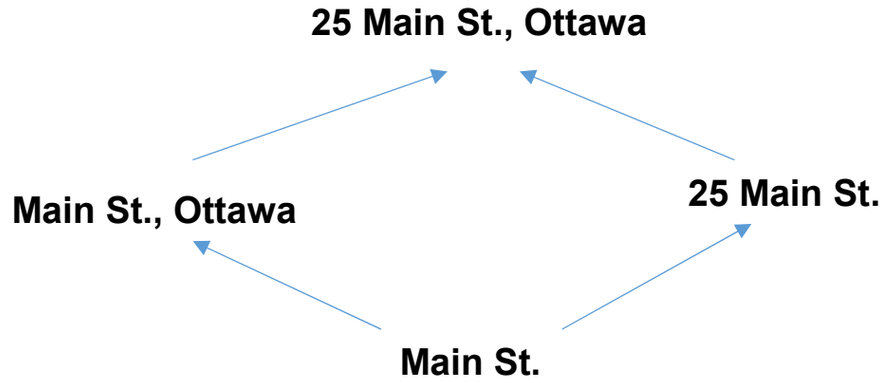
Addresses treated as strings or objects, i.e. sets of pairs attribute/value

- Matching functions induce a partial order (lattice) on attribute domains

$$a \preceq_A a' \iff \mathsf{m}_A(a, a') = a'$$

$a \preceq_A a'$ can be thought of in terms of information contents

When MFs are applied, information contents increases, and uncertainty decreases!



$D_0 \sqsubseteq D_1 \sqsubseteq \ldots \sqsubseteq D_{clean}$

- In general, there could be multiple clean instances

  Clean query answering (i.e. obtained from all clean instances) can be NP-complete

- For some special cases, among them:

  - Similarity-preserving matching functions

    $$a \approx a' \quad \Rightarrow \quad a \approx \mathsf{m}_A(a', a'')$$

  - Interaction-free MDs

  There is a unique clean instance D_{clean}

  It can be computed in polynomial-time in data

ERBlox

Joint work with: [IJAR 2017]

Zeinab Bahmani (Carleton University)

Nikolaos Vasiloglou (LogicBlox Inc.)

Entity Resolution

- A database may contain several representations of the same external entity

  The database has to be cleaned from duplicates

- The problem of entity resolution (ER) is about:

  (A) Detecting duplicates, as pairs of records or clusters thereof

  (B) Merging duplicates into single representations

- Much room for machine learning (ML) techniques
- We concentrate mostly on (A)

Blocking: Detecting Potential Duplicates

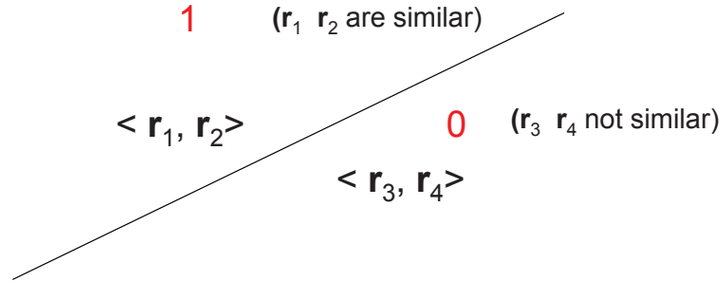
- We need to:

  (a) Compare pairs of records, for elements of a same entity (class):

  $$\mathbf{r_1} = \langle a_1, \ldots, a_n \rangle \quad \text{vs.} \quad \mathbf{r_2} = \langle a'_1, \ldots, a'_n \rangle$$

  (b) Discriminate between pairs of duplicate records and pairs of non-duplicate records

- A classification problem



- In principle, every two records have to be compared, and classified

  This can be costly ...

- Need to reduce the large amount of two-record comparisons

  ER mechanisms use blocking techniques

- A single attribute in records, or a combination of attributes, called a blocking key, is used to split records into blocks

  $\mathbf{r} = \langle \underline{a_1}, a_2, \ldots, \underline{a_5}, \ldots, \underline{a_8}, a_9 \rangle$ BK $= \langle A_1, A_5, A_8 \rangle$

  Only records within the same block values are compared

  Any two records in different blocks will never be duplicates

- For example, block employee records according to the city

  Compare only employee records with the same city

- After blocking many record-pairs that are clear non-duplicates are not further considered

  But true duplicate pairs may be missed

- For example, due to data input errors or typographical variations in attribute values

  Even assuming data is free of those problems, we need "similarity" functions:

  "Joseph Doe" and "Joe Doe" may not be errors, but possible different representations of the same:

  $$s_{name}(\text{"Joseph Doe"}, \text{"Joe Doe"}) = 0.9$$

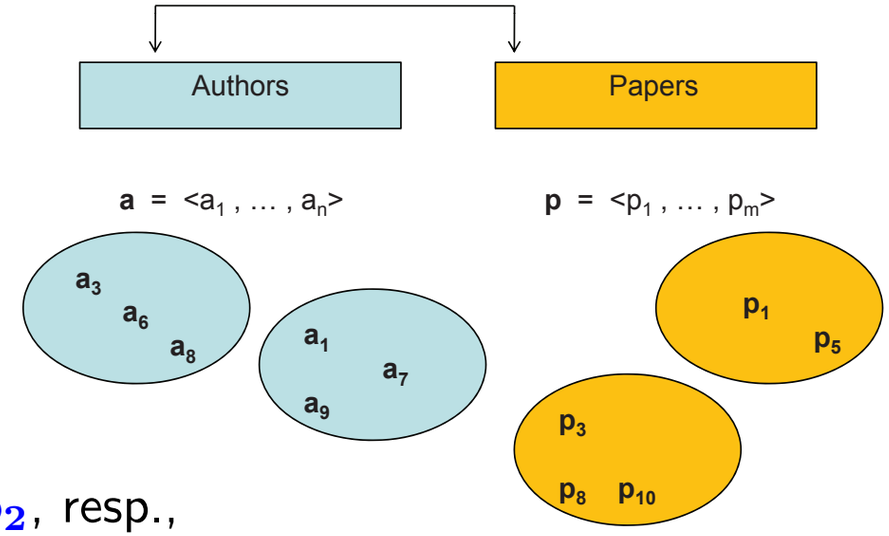
- So, now records in a same block have their BK attributes with "similar" values

- But still, grouping entities into blocks using just BK similarities may cause low recall

- It is useful to apply blocking with additional semantics and/or domain knowledge

Example: "author" and "paper" entities (records)

Want to group author entities based on similarities of authors' names and affiliations



Assume author entities $\mathbf{a_1}, \mathbf{a_2}$ (complete author records) have similar names, but not similar affiliations

$\mathbf{a_1}, \mathbf{a_2}$ are authors of papers (entities) $\mathbf{p_1}, \mathbf{p_2}$, resp.,

$\mathbf{p_1}, \mathbf{p_2}$ have been put in the same block of papers

Semantic knowledge: *"If two papers are in the same block, their authors with similar names should be in the same block"*

So, assign $\mathbf{a_1}, \mathbf{a_2}$ to same block (they could be duplicates)

- This is blocking of author and paper entities, separately, but collectively

  According to relational closeness, not only to similarities at attribute level

- How can we capture this kind of additional semantic knowledge?

  With a MD like this:

  $$Author(x_1, y_1, bl_1) \ \wedge \ Paper(y_1, z_1, bl_3) \ \wedge \ Author(x_2, y_2, bl_2) \wedge$$

  $$Paper(y_2, z_2, bl_3) \ \wedge x_1 \approx_1 x_2 \ \wedge \ z_1 \approx_2 z_2 \implies bl_1 \doteq bl_2$$

  This is (an extended form of) a matching dependency (MD), used here for blocking

  Originally for merging attribute values, not for blocking

- ML could be used to create the blocks, e.g. using clustering methods (part of 1st ER phase)

  Not what we do here ...

- We use MDs for blocking, before the ML-based classification task

MD-Based Collective Blocking

- Records have unique, global ids (positive integer values)

  Initial block number $Bl\#$ for a record is its rid

- Two records are forced to go into same block by enforcing the equality of their block numbers

  Use MDs with a MF: $\mathsf{m}_{Bl\#}(b_i, b_j) := b_i$ if $b_j \leq b_i$

Example: Author and Paper entities ("$R.\ Smith$" $\approx$ "$MR.\ Smyth$")

Author	*Name*	*Affiliation*	*PaperID*	*Bl#*
12	*R. Smith*	*MBA, UCLA*	1	12
13	*MR. Smyth*	*MBA*	2	13
14	*J. Doe*	*MBA, UCLA*	3	14

Paper	*Title*	*Year*	*AuthorID*	*Bl#*
1	*Illness in Africa*	1990	12	2
2	*Illness in West Africa*	90	13	2

"Group two author entities into same block if they have similar names and affiliations or they have similar names and their corresponding papers are in same block"

$$m_1: \quad Author(a_1, x_1, y_1, p_1, b_1) \wedge Author(a_2, x_2, y_2, p_2, b_2) \wedge x_1 \approx x_2 \ \wedge \ y_1 \approx y_2 \ \Rightarrow \ b_1 \doteq b_2$$

$$m_2: \quad Author(a_1, x_1, y_1, p_1, b_1) \ \wedge Author(a_2, x_2, y_2, p_2, b_2) \ \wedge x_1 \approx x_2 \wedge Paper(p_1, x_1', y_1', a_1, b_3) \ \wedge \ Paper(p_2, x_2', y_2', a_2, b_3) \ \Rightarrow \ b_1 \doteq b_2$$

First is a single-entity blocking MD

The second is an inter-entity blocking MD

Applying them results in a DB instance with two author-blocks: $\{12, 13\}, \ \{14\}$

- Here we are supporting ML with the use of semantic knowledge

  One can go beyond that ...

- A newer research area is about developing ML methods that involve semantic knowledge

  As expressed by logical formulas

  Room for Statistical Relational Learning ...

- After blocking we may start classifying pairs

Classifying Record-Pairs (general)

- ML techniques are commonly used to discriminate between:
  - pairs of duplicate records (of each other), i.e. duplicate pairs, and
  - pairs of non-duplicate records, i.e. non-duplicate pairs
- ML is used here to classify record-pairs

  (still part of 1st phase of ER)
- We developed a classification model

  The classification hyper-plane on slide 21 ...

All this is part of the *ERBlox* approach/system

The *ERBlox* Approach to ER

- *ERBlox* enables/supports ML-techniques for ER

  Different ML techniques can be used for the classification model

  ER is based on supervised ML techniques, which require training data

  We used "support-vector machine" (SVM)

- *ERBlox* also based on the use of MDs

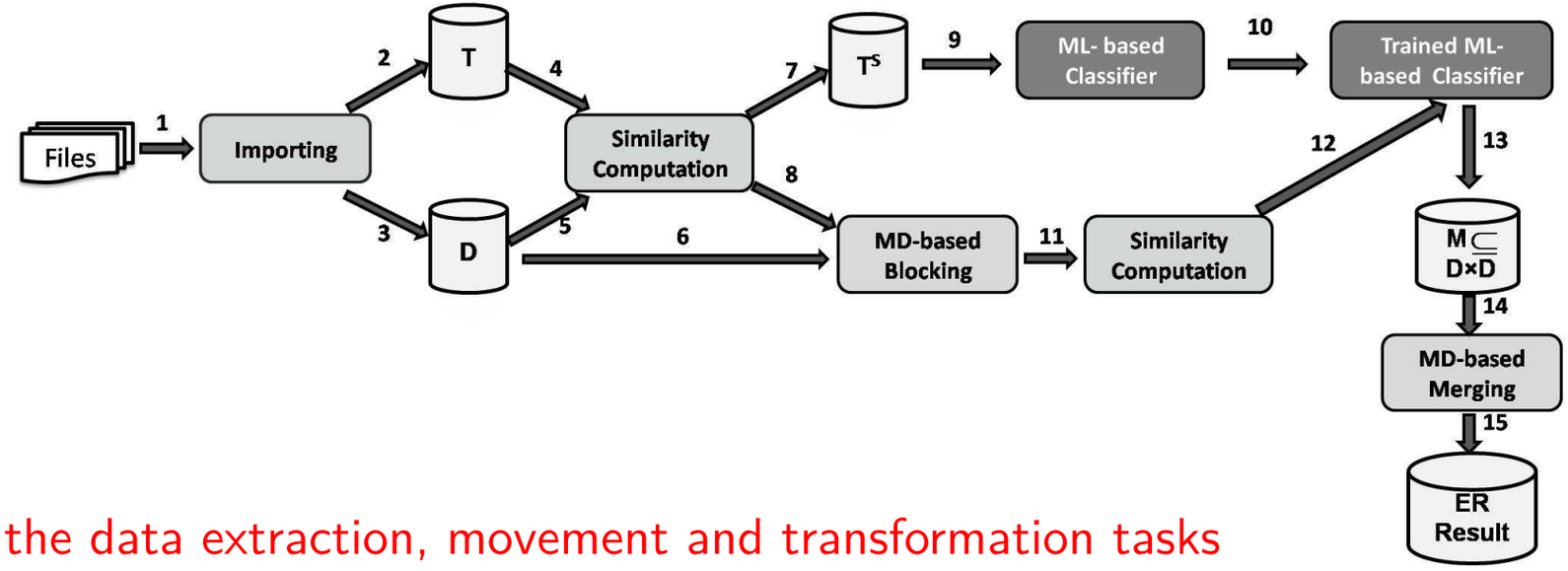
- *ERBlox* interacts with Datalog-based relational DBs

  Profiting from Datalog for different tasks

  More specifically, the *LogicBlox* system (*LogiQL*, now)

- *ERBlox* has three main components:

  1. MD-based collective blocking
  2. ML-based record duplicate detection
  3. MD-based merging



- All the data extraction, movement and transformation tasks carried out via *LogicQL*'s extended Datalog

Merging

- After the classification task, (records in) duplicate-pairs have to be merged

  Records in them are considered to be "similar"

  In a precise mathematical sense, through the use of domain- dependent "features"

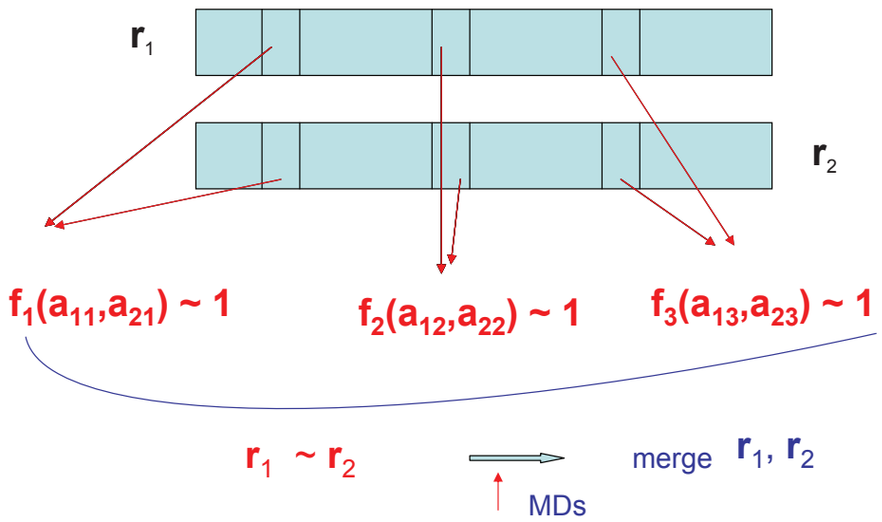
  MDs are also used for merging (their common use)

- Different sets of MDs for blocking and merging

- The classifier decides if records $\mathbf{r_1}, \mathbf{r_2}$ are duplicates or not

  Resp. returning $\langle \mathbf{r_1}, \mathbf{r_2}, 1 \rangle$ or $\langle \mathbf{r_1}, \mathbf{r_2}, 0 \rangle$

In the first case this means values for some features on pairs of attribute values are close to 1



- Define: $r_1 \sim r_2 \ :\Longleftrightarrow\ \langle r_1, r_2, 1\rangle$ is output
- Merge-MDs of the form: $r_1 \sim r_2 \implies r_1 \doteq r_2$

  LHS means $\langle r_1, r_2\rangle$ is given value 1 by classifier

  RHS means $r_1[A_1] \doteq r_2[A_1] \wedge \cdots \wedge r_1[A_m] \doteq r_2[A_m]$
- Mergings on RHS, based on domain-dependent matching functions (MFs)

MD-Based Merging

- We enforce MDs to merge duplicate records into single representations
- We consider record-level MDs:

$$\varphi\colon \quad R[t_1] \approx R[t_2] \implies R[\bar{Z}_1] \doteq R[\bar{Z}_2]$$

$\bar{Z}_1$, $\bar{Z}_2$ contain all attributes of R

- The MDs implicitly contain all attributes on the LHS

  The LHSs imposes the condition that two tuples are duplicates (a higher-level notion of similarity)

  Its truth is evaluated according to the output of the classifier

Example: Merge duplicate author-records enforcing the MD:

$$Author[aid_1] \approx Author[aid_2] \implies$$
$$Author[Name, Affiliation, PaperID] \doteq Author[Name, Affiliation, PaperID]$$

(LHS abbreviation for $Author \sim Author$)

- A derived table $Author\text{-}Duplicate$ is used on LHS, with contents computed before merging and kept fixed during the enforcement of merge-MDs

  In this way, transitivity of record similarity is captured ...

  This makes the sets of merging-MDs interaction-free

  Resulting in a unique resolved instance

  (similarly for enforcement of blocking-MDs)

On the Use of MDs

- In general, application of MDs on an instance may produce alternative, admissible instances

- General MDs can be specified/enforced with answer-set programs (ASPs)
  [Bahmani et al., KR'12]

  General ASP not supported by *LogiQL*

- We obtain a single blocking solution, applying "blocking MDs"

- On that basis, also the final result of ER is a single duplicate- free instance, applying "merge-MDs"

- The kind of MDs in our case, and the way there are use/applied, requires only "stratified Datalog", which is supported by *LogiQL*

  Our MDs can be specified/executed with LogiQL's Datalog

Discussion:
Some New Trends in Data Science

- *ERBlox* developed in collaboration with LogicBlox Inc. ($\rightarrow$ Relational^{AI})
  `http://www.logicblox.com/`

  It is built on top of the *LogicBlox* Datalog platform (LogiQL)

- Goal: Extend LogiQL extending, implementing and leveraging Datalog technology

- Datalog enables declarative and executable specifications of data-related tasks

  An extension of relational algebra/calculus/databases

  A query language and view definition language for relational DBs

- Datalog has been around since the early 80s

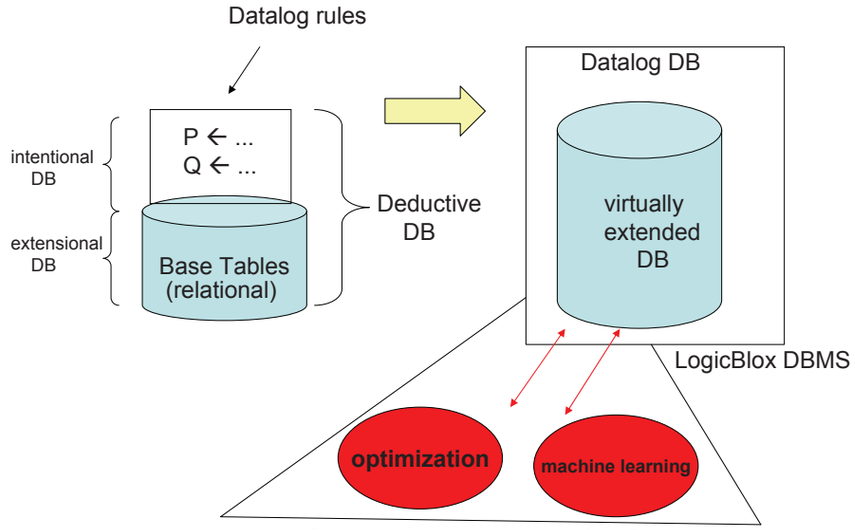
  Strong theoretical basis, prototype implementations, used mostly in DB research (then)

  Datalog techniques became part of the SQL standards and commercial DBMSs

- Datalog has experienced a revival during the last few years

  Many new applications have been found!

- LogicQL is being extended with interaction with optimization and ML packages and systems!



mixed-integer (linear) programming, regression (classical, logistic), SVMs, ...

- Data for these problems stored as "extensions" for DB & Datalog predicates

  ML system reads necessary data from tables or Datalog computations

  ML results become contents for newly defined predicates

  Smooth interaction between Datalog/relational engine and optimization/ML

- For this to work efficiently in practice: Implement the fastest underlying DBMS!

  In particular the fastest data combination (join) algorithms

- LogiQL has implementations of the provably optimal join algorithms!

- What just mentioned above already largely outdated:

  New trend: In-database computations!

  Use the DBMS engine to do the ML/Optimization-related computations

  Push computations inside the DB!

- Along the same line, a new research area: Do linear algebra inside the DB

  Matrices and tensors represent data

  Develop LA-oriented query languages

  E.g. a query asking for the (first) eigen values of a matrix

- All these recent developments heavily affect research and practice of data science

  Much to do in the intersection/interaction area of data management and AI

- We are living exciting times!

EXTRA SLIDES

Conditional Dependencies (CDs)

Example: Database relation with FDs:

FD_1: $[CC, AC, Phone] \rightarrow [Street, City, Zip]$

FD_2: $[CC, AC] \rightarrow [City]$

CC	AC	Phone	Name	Street	City	Zip
44	131	1234567	mike	mayfield	NYC	EH4 8LE
44	131	3456789	rick	crichton	NYC	EH4 8LE
01	908	3456789	joe	mtn ave	NYC	07974

FDs are satisfied, but they are "global" ICs

They may not capture natural data quality requirements ...

... those related to specific data values

- What about a *conditional functional dependency* (CFD)?

$$CFD_1: \quad [CC = 44, Zip] \rightarrow [Street]$$

The FD of $Street$ upon Zip applies when the country code is 44

Not satisfied anymore, and data cleaning may be necessary ...

- More generally, CDs are like classical ICs with a *tableau* for forced data value associations

CFD_2:

$$[CC = 44, AC = 131, Phone] \rightarrow [Street, City = \text{`}EDI', Zip]$$

When $CC = 44, AC = 131$ hold, the FD of $Street$ and Zip upon $Phone$ applies, and the city is 'EDI'

Not satisfied either ...

- CQA and database repairs have been investigated for CFDs
- We can go one step further ...
- Conditional Inclusion Dependencies:

$$Order(Title, Price, Type = \text{`}book') \subseteq Book(Title, Price)$$

It can be expressed in classical FO predicate logic:

$$\forall x \forall y \forall z (Order(x, y, z) \wedge z = \text{`}book' \rightarrow Book(x, y))$$

Still a classic flavor ...

And semantics ...

Experimental Evaluation

- We experimented with our *ERBlox* system using datasets of Microsoft Academic Search (MAS), DBLP and Cora

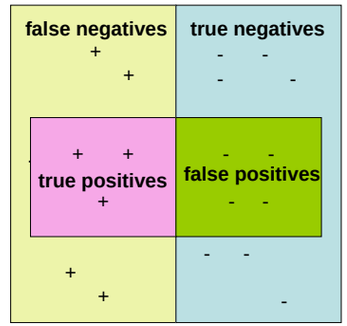
  MAS (as of January 2013) includes 250K authors and 2.5M papers, and a training set

- We used two other classification methods in addition to SVM

- The experimental results show that our system improves ER accuracy over traditional blocking techniques where just blocking-key similarities are used

- Actually, MD-based collective blocking leads to higher precision and recall on the given datasets

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{Total}}$$



precision = true positives / (true positives + false positives)

recall = true positives / (true positives + false negatives)