# Efficiently Identifying Disguised Missing Values in Heterogeneous, Text-Rich Data

Théo Bouganim, Ioana Manolescu, Helena Galhardas

# Efficiently identifying disguised missing values in heterogeneous, text-rich data

Théo Bouganim theo.bouganim@inria.fr[1,2], Helena Galhardas hig@inesc-id.pt[3], and Ioana Manolescu ioana.manolescu@inria.fr[1,2]

[1] Inria Saclay Ile de France, Palaiseau 91120, France
[2] Institut Polytechnique de Paris (IPP)
[3] INESC-ID & IST, Universidade de Lisboa, Portugal

**Abstract.** Digital data is produced in many data models, ranging from highly structured (typically relational) to semi-structured models (XML, JSON) to various graph formats (RDF, property graphs) or text. Most real-world datasets contain a certain amount of *null* values, denoting missing, unknown, or inapplicable information. While some data models allow representing nulls by special tokens, so-called *disguised missing values (DMVs, in short)* are also frequently encountered: these are values that are not syntactically speaking nulls, but which do, nevertheless, denote the absence, unavailability, or inapplicability of the information. In this work, we tackle the detection of a particular kind of DMV: texts freely entered by human users. This problem is not tackled by DMV detection methods focused on numeric or categoric data; further, it also escapes DMV detection methods based on value frequency, since such free texts are often different from each other, thus most DMVs are unique. We encountered this problem within the ConnectionLens [6,7,8,12] project where heterogeneous data is integrated into large graphs. We present two DMV detection methods for our specific problem: (*i*) leveraging Information Extraction, already applied in ConnectionLens graphs; and (*ii*) through text embeddings and classification. We detail their performance-precision trade-offs on real-world datasets.

## 1 Introduction

Digital data is being produced and reused at unprecedented rates. Large datasets are usually processed within data management systems, which *model* the data according to a given data model, provide means to *store* it, and *query* it using a query language. The database industry has been pioneered by Relational Database Management Systems (RDBMSs), whose foundations lay in first-order logic, formalization by E. F. Codd [13], and subsequent work, e.g., [3].

**Where there is data, there are null values** Since the early database days, *nulls* have been identified as a central concept denoting missing, unknown, or inapplicable information. The semi-structured data model first embodied in OEM (the Object Exchange Model), proposed to simply omit missing values from the

data [17]. However, standard semi-structured models such as XML and JSON re-introduced nulls, e.g., xsi:nil in tools supporting XML Schema or the special null value in JSON. Such null tokens may have been needed in practice because perfect, complete databases, regardless of their data model, are the exception rather than the norm.

**Disguised missing values** In practice, relational databases often feature not only *null* tokens but also *non-null values playing the semantic role of nulls*, also called *disguised missing values (DMVs, in short)* [2]. For instance, 0 or $-1$ are often used to encode an unknown (but non-zero) number such as a price; users may enter "none, "-", "unknown" or "N/A" or any other similar phrase or token, in entry forms requiring numbers, names or dates that they are unable or unwilling to fill in. Further, when a value needs to be chosen from a predefined set, such as a state of the U.S., users may forget to set it from the menu, leaving the default value which just happens to be the first, e.g., "Alabama" as a U.S. state.

Data entry forms sometimes prevent DMVs by checking the entered value, e.g., "N/A" would not be accepted as a number. However, DMVs may still persist: (*i*) users replace "N/A" with 0 for an unknown, non-zero number; (*ii*) if the expected input type is free text, e.g., "List of industrial collaborations in connection with this research", no simple format-driven validation applies; (*iii*) in cases such as "Alabama" above, the value is in the correct domain.

**Detecting DMVs** Identifying DMVs requires dedicated methods, and several have been proposed for relational databases [19,23,24,25], based on a *statistical analysis* of the data. They can detect, for instance, when a value such as 0 is suspiciously frequent in a numeric attribute, or when a value of attribute $R.a$ is an outlier in the joint distribution of $(R.a, R.b)$, where we expect the distributions $R.a$, $R.b$ to be independent. More approaches, in particular *rule-based*, are discussed in [1]. Other works also propose corrections for erroneous or missing values, e.g., [22]. Detecting and correcting DMVs is important for data cleaning (users may want to replace them with explicit nulls), and for query correctness: null values should not match any selection predicate, and there should be no join on null values.

**Problem statement and outline** In this work, we consider the detection of DMVs in *textual, heterogeneous data*. The motivation for our work came from ConnectionLens [7,8,12], a system that integrates structured, semi-structured, or unstructured data into graphs, enriched by adding all the entities (people, organizations, places, URIs, dates, etc.) encountered in various text nodes.

We have developed ConnectionLens inspired by fact-checking and data journalism applications [6,9]. In such applications, we encountered many datasets where some fields are *free-form text* entered by users. For instance, in the French Transparency dataset HATVP[4], elected officials need to state "their direct finan-

[4] https://hatvp.fr

cial participation in company capitals"; in the PubMed bibliographic database[5], free-form texts include the article titles, abstracts, funding statements, and possible acknowledgments. Detecting DMVs in ConnectionLens graphs is interesting from the following perspective: if we know that a string is a DMV, we can avoid extracting entities from it, thus reducing the time needed to construct ConnectionLens graphs [6,7].

DMVs encountered in free-form text fields can be classified into two broad classes:

1. Short, simple strings such as "N/A" or "-", which tend to be frequent, and thus can be detected using previously proposed methods, such as [25,23];
2. Complex phrases such as "Liz Smith has not received any funding related to this work", "No conflicts of interest, financial, or otherwise are declared by the authors", or "There is no conflict of interest relating to Authors. The manuscript was prepared according to scientific and ethical rules".

To detect DMVs of the second form above, our work is organized as follows:

1. We recall the closest existing DMV detection techniques (Section 2), then we introduce a motivating example to support our discussion (Section 3).
2. We show that ConnectionLens Named Entity Recognition can be leveraged to (manually) establish an *entity profile* for each set of text attributes in which we want to detect DMVs and consider any value deviating from this profile a DMV. For instance, the entity profile of financial participation descriptions could be *Organization+* to state that it should contain at least one organization. Profiles allow defining, in semistructured and heterogeneous graphs, groups of values on which we can reason about DMVs. This method is quite accurate, however, it incurs a high computational cost, since entity extraction is a complex operation (Section 4).
3. To address this shortcoming, we devised a novel method, which relies on *text embeddings and classification*, while also leveraging entity extraction on a much smaller portion of the dataset. This method is much more efficient than the one based on entity profiles, while also being very accurate (Section 5).
4. We show how we *integrated this novel method within the architecture of ConnectionLens* (Section 6) to speed up graph construction.
5. In our experimental evaluation (Section 7): (*i*) We perform a set of experiments on state of the art methods, and show that they do not perform well on the DMVs we target in this work. (*ii*) We study the efficiency and precision of our method based on embeddings and classification. (*iii*) We experimentally validate the interest of including it within the ConnectionLens system, demonstrating that it reduces the graph construction time.

This work is an invited extension of a short, informally presented paper [10]. A core contribution of this paper with respect to that prior work is the integration of our DMV detection method within ConnectionLens (Section 6), together with

[5] https://pubmed.ncbi.nlm.nih.gov/

the associated experiments (Section 7.7) validating the practical interest of this method. Other novel extensions, beyond improving and clarifying the writing, include comparisons with a method based on sentence-Bert [27] (Section 7.5), and a validation of the quality and performance of our methods on a manually labeled dataset (Section 7.6).

## 2 Related Work

In this section, we recall the main DMV detection methods, as well as some closely related efforts.

Foundational work in this area was made in [24], which introduced and formalized the problem of Disguised Missing Values (DMV), and measured its influence on different data science models. A set of *statistical models* (mutually disjoint hypothesis) were introduced in [21] to model nulls as well as disguised missing values:

- The **MCAR** (Missing Completely At Random) model posits that the probability of a value to be missing is the same for any value of an attribute, and does not depend on the values of any other attribute. For instance, assuming an attribute is the result of a physical measure made with a device that breaks down, the resulting missing values are not correlated to any other aspect of the data or of the values.
- The **MAR** (Missing At Random) model considers that the probability for a value being missing depends on values encountered in other attributes of the same table (these notions have been defined for tables). For example, in a political poll, assume *young* voters are more likely not to declare their political preference. Then, the political preference value is MAR.
- **MNAR** (Missing Not At Random) applies when neither MAR nor MCAR holds, and when the probability for a value to be missing does depend on the actual value that is missing, but not on the values of any other attribute. For instance, assuming supporters of a certain political party generally avoid stating their preference and instead let that information go missing, such values are MNAR.

Building upon these models, [19] has proposed a heuristic method for identifying DMVs in relational databases. Under the MAR and MCAR assumptions, the authors assume that a value $v$ in attribute $A_i$ in a table $T$ is a DMV if $\sigma_{A_i=v}(T)$ contains a subset $T^*_{A_i=v}$ that represents a good sampling of $T$. Such a subset is an *Embedded Unbiased Sample* (**EUS**) which means that except for attribute $A_i$, $T^*_{A_i=v}$ and $T$ have similar distributions. Then, a **MEUS** (*Maximal EUS*) is intuitively an EUS with a good trade-off between size (larger is better) and similarity (in distribution) with $T$. Thus, the MEUS is the largest EUS with the highest similarity. The gist of the [19] heuristics is to find MEUS in a dataset and consider their associated $A_i = v$ values as DMVs.

FAHES [25] incorporates the method of [19], to which the authors add two other methods, in order to distinguish three classes of DMV.

- The first class contains **syntactic outliers**. A syntactic outlier is a value whose syntax is significantly different from that of other values in the same attribute. Two techniques are used to identify them. (*i*) Syntactic pattern discovery infers a frequent syntactic pattern (shape) for the values of each attribute and points out the values that do not fit the pattern as syntactic outliers. For example, if the attribute is "blood type", the recognized pattern could be *one or two uppercase letters followed by a + or a - sign*; then, "ABO" would be considered a syntactic outlier. (*ii*) Repeated Pattern Identification singles out values that contain repeated patterns, such as 0101010101 in a 10-digit phone number, or "blablabla" in a text attribute.
- Second, in numerical attributes, **statistical outliers** can be found by leveraging common outlier detection methods [18]. This allows identifying as DMVs numerical values that do not fit the extent of the other values, e.g., negative values in a distance attribute.
- Finally, some **inlier DMVs**, called **Random DMVs** in [25] can be identified. These are legal attribute values, which do not stand out as outliers; they are the hardest to find even for an application domain specialist. The "Alabama" example from Section 1 is a typical example. Inlier DMVs are detected in [25] under the MAR and MCAR hypotheses; the authors state that detecting DMVs under the NMAR model is hard to impossible. The intuition being exploited is that *DMVs are frequent values* (because the lack of information is assumed to occur more frequently than an actual, correct value). Thus, one must find amongst the most frequent values, which ones are DMVs. To this purpose, each frequent value is successively replaced by an actual null. If, by doing this, the (original and introduced) null values follow the MAR or MCAR models, then we consider that value as a good DMV candidate. Then, the MEUS method from [19] is applied to each candidate to detect the DMVs.

The FAHES team has developed a tool using these methods to detect all three types of DMV in a relational database.

DMV detection is also related to *data error detection and correction*, which has been studied in [1]. A tool called RAHA [23] has been developed for detecting errors in relational databases; it detects the DMVs that FAHES [25] finds, as well as errors that are not DMVs. BARAN [22] corrects data errors through a combination of multiple techniques. The free-text DMVs we are interested in are not errors, and their values should be preserved as such. From this perspective, FAHES [25] is closest to our DMV detection goal.

DMV detection is one among the many problems raised by poor *data quality* problems that have been traditionally addressed through *data cleaning*. Data quality raises many real problems, which to this day still need solutions. Traditional approaches for data cleaning were rule-based [15,16,26]. Newer techniques are now based on machine learning, e.g., [20]. DMV detection can also be considered as part of data profiling [2] since DMV detection also allows the characterization of a certain attribute (set of nodes) by the percentage of their values which are DMVs.

Summing up, the existing literature has proposed methods for detecting DMVs and errors, in categorical or text data, with rule-based and especially statistical and more recently machine learning techniques. Our focus is on DMV occurring in free-text data, which are not detected by methods based on value frequency, as we illustrate below.

## 3 Motivating example

As part of a data journalism project [6,8], we loaded 400.000 **PubMed bibliographic notices** in a ConnectionLens graph, out of which we extracted (paper ID, **conflict of interest** statement) pairs. These conflict of interest (**CoI**, in short) statements cover any kind of benefits (funding, personal fees, etc.) that authors report with various organizations such as companies, foundations, etc.

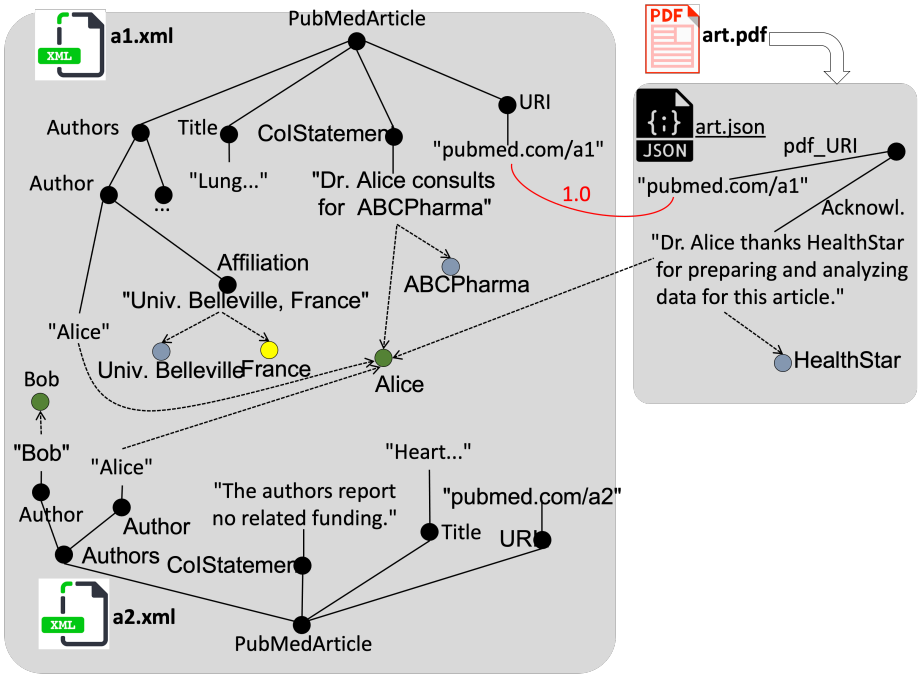


**Fig. 1.** Sample ConnectionLens graph.

In Figure 1, "Dr. Alice consults for ABCPharma" (in the upper left) is such a conflict of interest, part of the XML bibliographic notice; "Dr. Alice thanks HealthStar... this article" (at the top right) is another one. "The authors report no related funding" in the second paper, at the bottom of the figure, illustrates the DMVs targeted in this work. PubMed data originates from various biomedical journals. Some do not provide CoI information; in this case, the CoI is an

empty string. Others provide a default DMV, e.g., "The authors report no conflict". Finally, some journals only allow free text, leading to a large variety of disguised nulls.

ConnectionLens extracts **named entities** from all text nodes, regardless of the data source they come from, using trained language models. In the figure, blue, green, and yellow nodes denote Organization, Person, and Location entities, respectively. Each entity node is connected to the text node it has been extracted from, by an extraction edge, which also records the confidence (between 0 and 1) of the extractor. Finally, nodes are compared to find that some may be equivalent, or similar.

Entity extraction is a costly operation since it involves predicting, for each token encountered in a given text, if it is part of an Organization entity, Person entity, Location entity, or none. The prediction is made by computations that involve a large, trained model. Trying to extract entities from a DMV is a computational effort spent for no benefit.

## 4 Detecting DMVs with entity profiles

Inspecting the ConnectionLens graph illustrated in Figure 1, together with our journalist partner, we immediately made the following observation. An actual CoI (such as those involving Alice in Figure 1) is either of the form "Researcher A was funded by B", or of the form "The authors acknowledge funding from C". Thus, a person's name may be present (in other cases, we just find "The authors"), but an organization is always involved. Thus, we can say that the *entity profile* of a CoI is: it must contain at least an organization.

This leads to the following DMV detection method:

- Extract all named entities from the CoI strings (through regular ConnectionLens data ingestion);
- Declare those CoI strings in which no Organization entity was found, as DMVs.

One could also call such DMVs *uninformative answers*. **We use the result of the above entity-based DMV detection method as a ground truth in our work**, for several reasons: (*i*) Named Entity Recognition is by now a relatively well-mastered task, thus its precision is quite good, and considered acceptable by our end-users; (*ii*) Constructing an ideal human-authored ground truth would require time or monetary costs out of reach for our setting; (*iii*) The very purpose of our work is to save Named Entity extraction time, in other words: Named Entity extraction at the core of ConnectionLens' integration is a given in our context.

The accuracy of this method is exactly that of the entity extractor; it has been shown in [4,5] (for English) and in [7] (for French) that the accuracy is quite high. Its drawback is that *extracting entities from all CoI strings is very lengthy*.

This motivates the search for a faster technique, which, on one hand, could identify the DMVs, while at the same time also reducing the entity extraction (thus, the actual ConnectionLens graph creation) time.

## 5 DMV detection through embedding and classification

Our initial DMV detection approach, which does not require extracting named entities from all strings, has been to *cluster* the text values from our motivating example dataset, in order to obtain DMV cluster(s) separated from non-DMV clusters. In particular, we experimented with the K-means [18] algorithm, setting the number of clusters to 10 which was the optimal number of clusters determined using the elbow method[6]. Some clusters contained a majority of DMVs and others a majority of CoIs. However, the clustering was not very successful at separating DMVs from meaningful CoIs.

Thus, we looked for an alternative method. Our idea is: we could extract entities from a small part of the data, so we have a small *automatically* labeled dataset to train a Machine Learning model to recognize DMVs (based on the method described in Section 4). We could then use this model to predict whether a yet-unseen value is a DMV or not. Such a model could detect DMVs faster than using the entity extraction technique.

### 5.1 Textual data representation

Many techniques can be used to represent textual data. Transformers like BERT [14] have been proven really efficient for many Natural Language Processing (NLP) tasks; Sentence-BERT [27] provides sentence-level embeddings.

An alternative, less costly textual data representation can be computed by first applying a set of common text pre-processing steps: suppressing punctuation, normalization, and stemming. Embeddings can then be computed using the well-known TF-IDF (Term-Frequency - Inverse-Document-Frequency) representation, also frequently used in NLP. TF-IDF weights term frequencies in each document according to the frequency of the term across the corpus. If a word occurs many times in a document, its relevance is boosted (TF part of the score), as this word is likely to be more meaningful. Conversely, IDF stands for the fact that if the word appears frequently in many documents, then it is probably frequent in any text, and its relevance should be decreased. Finally, to reduce the dimensionality of the representation, we only consider the terms having the 20.000 highest TF-IDF scores.

We will compare Sentence-BERT representation with TF-IDF representation for this task of detecting DMVs in free-form texts. However, we integrate into ConnectionLens (Section 6.2) the TF-IDF version, because it is faster yet provides equivalent quality.

[6] https://en.wikipedia.org/wiki/Elbow_method_(clustering)

### 5.2 Classification model

To classify texts as DMVs or non-DMVs, we decided to rely on a Random Forests classifier [11]. These classifiers are not the fastest, but they are quite efficient over complex data. Random Forests rely on decision trees, which seemed appropriate in our context, as they could learn specific discriminating words that help to differentiate the classes. Other classifiers might work as well; our goal here is to investigate whether the above approach, which trains the classifier on extraction results, can provide a more efficient DMV detection method, by avoiding extracting entities from a certain part of the input.

As we show in Section 7.5, this is indeed the case; even for small training set sizes (that is, even if entities are fully extracted only from a small part of the data), the classifier learns to predict DMVs quite accurately, while sparing significant entity extraction time comparing to the entity profile method.

### 5.3 Placing our DMV detection methods in context

To further clarify the relationship between our work and prior DMV detection work, Figure 2 depicts known types of DMVs as rectangles carrying white titles, and the values which various techniques find to be DMVs, as ovals with black titles. As no method is perfect, rectangles and ovals only partially overlap; further, some DMVs are detected by more than one method, and thus some ovals overlap. We assigned numbers to some areas in the figure, and comment on them below. It helps to keep in mind that an area in a rectangle but outside of an oval comprises DMVs that the method corresponding to the oval does not detect; conversely, an area within an oval but outside of the DMV rectangle(s) is a declared by the method to be a DMV, while it is not.

1. Correctly detected statistical outliers, e.g.: In a human height attribute, a height of 3 meters.
2. Wrongly detected statistical outliers, e.g.: In a dataset containing salaries of the employees of a company, if the CEO's salary is 10 times higher than any of his employees, this could be wrongly detected as a DMV.
3. Correctly detected syntactic outliers, e.g.: In a blood type attribute, the value *'ABO'*.
4. Wrongly detected syntactic outliers, e.g.: In a name attribute, *François-Noël* which is a composed name is detected as a syntactic outlier because of the *'-'*, even though it is a valid name.
5. Correctly detected Random DMVs, e.g.: A default value such as *Alabama* for a state, detected thanks to the MEUS technique.
6. Correctly detected Random DMVs found by the MEUS technique, and also by syntactic and statistic outliers detections. A DMV can be at the same time detected as a syntactic outlier, a random DMV, and a statistical outlier, e.g., a default distance value of -1.
7. Wrongly detected random DMVs. In a poll where we ask for favorite colors, *blue* might come often. Lacking correlation with other attributes, *blue* could be wrongly detected as a random DMV.

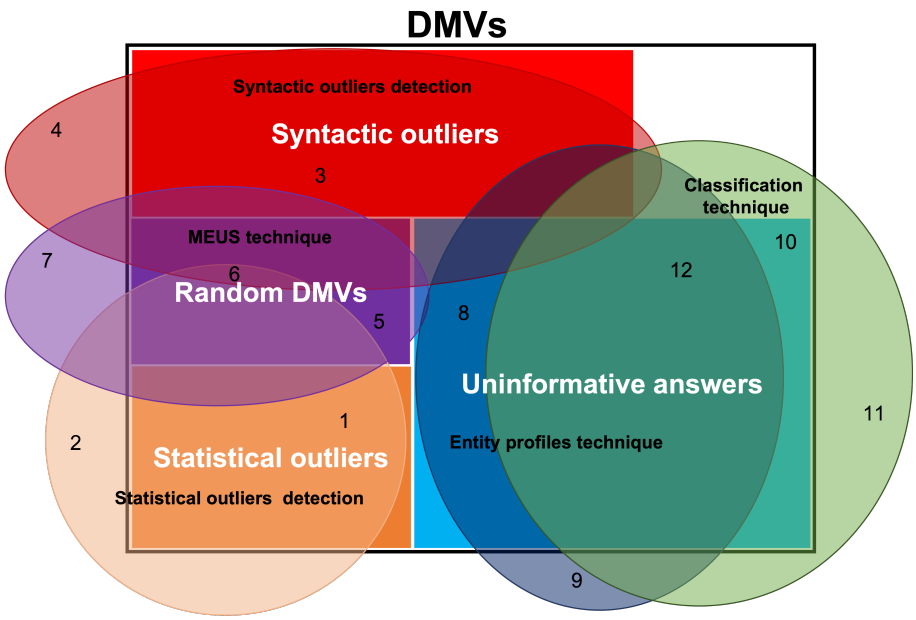


**Fig. 2.** DMVs and the scope of each DMV detection technique.

8. Correctly detected DMV with entity profile technique, e.g.: In a conflict of interest paragraph: *"The authors report no conflict of interest."*
9. Wrongly detected DMV with entity profile technique. These are errors of the entity extractor when it misses an organization.
10. Correctly detected DMV with the classification method that was not detected by the entity extraction method. For example, in *'John McDonalds declares no conflict of interest'*, the entity extractor could detect McDonalds as an Organisation, and classify the value as an actual CoI instead of a DMV.
11. Wrongly detected DMV with the classification method, these are errors of the classifier.
12. Correctly detected DMV with both classification and entity profile technique. Most of the DMVs detected by the entity profile technique are as well detected by the classification technique.

With respect to the the diagram in Figure 2, we *target* the Uninformative answers rectangle, with *two methods*: the Entity profile one, corresponding to the blue oval that encompasses the areas numbered 8, 9, and 12; and the Classification one, corresponding to the green oval, that comprises the areas numbered 10, 11 and 12. The closest method from the literature, FAHES, excels in finding: Syntactic outliers, Random DMVs, and Statistical outliers.

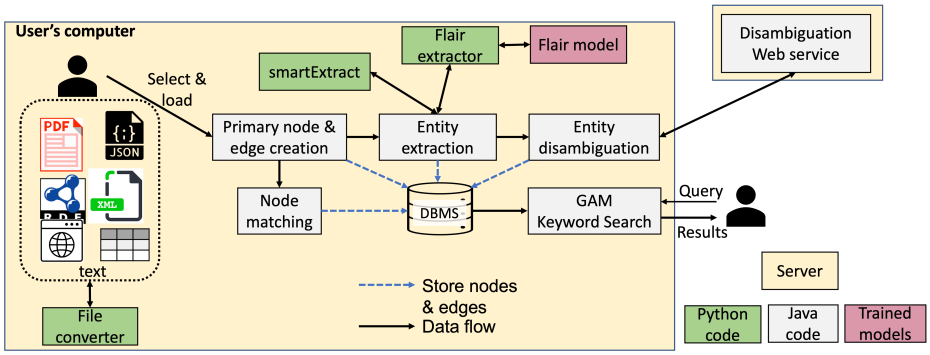


**Fig. 3.** ConnectionLens architecture (based on [7]).

## 6 Integrating our DMV detection methods within ConnectionLens

We recall the architecture of the ConnectionLens system (Section 6.1), then discuss how we integrated our DMV detection module within it (Section 6.2).

### 6.1 ConnectionLens architecture

The architecture of the ConnectionLens software platform is outlined in Figure 3. To consolidate heterogeneous data sources into a single graph, the sources are traversed, *creating primary nodes and edges* which represent the source contents (black solid nodes and edges in Figure 1). On the fly, also during the data source traversal, all the text values encountered in the data are sent to an *entity extraction* module, which may recognize named entities within these texts. Extracted entities are shown as colored nodes in Figure 1, and edges connecting them to their parent nodes are called *secondary edges*. ConnectionLens' entity extraction (for French and English) is based on Flair [4]. The cost of an extraction operation is relatively high compared with other kinds of processing taking place in memory on a data item, and quite high also when compared to the cost of storing data persistently on disk [7].

Within the ConnectionLens platform, the Flair entity extractor is deployed as a **Python Web service** using the **Flask** web service library; we were able to experimentally check that the performance overhead of calls from the main Java software to the Python web service was negligible. Depending on how many text nodes a data source contains, and how long they are, entity extraction may take a very large part of the graph construction time. A method we implemented to alleviate to some extent speeds up entity extraction by exploiting the parallel processing (multi-core) capabilities of the server on which the extraction runs. Concretely, a **batch** mechanism is implemented: text nodes accumulate in a fixed-size batch and the Flask service is called when the batch is full. It is more efficient for a multi-core machine to apply its prediction model to a set of values

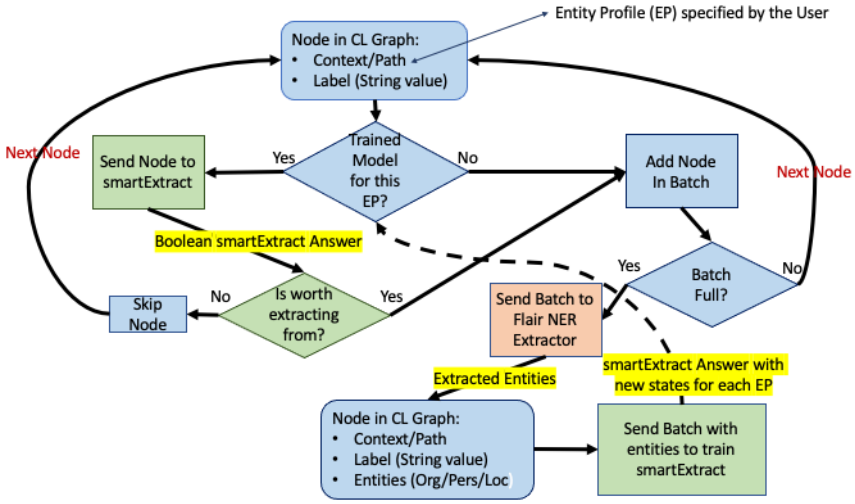


**Fig. 4.** Integration of DMV detection module within ConnectionLens.

in parallel, as enabled by sending them in batches; the benefits range from 2× on a 4-cores laptop to 20× on a powerful server [7]. However, entity extraction costs remain quite significant, and in some cases, they dominate the ConnectionLens graph construction time.

The remaining ConnectionLens modules are: a *File converter*, that allows ingesting document formats such as PDF, Word, Excel, etc. by converting them to JSON HTML, and/or RDF, which can be directly loaded; a *Node matching* module that creates equivalence or similarity links between entities (red, respectively, dashed graph edges in Figure 1); an *Entity disambiguation* service, which, for each entity identified in the graph, attempts to find its URI in a knowledge base; finally, a *Keyword search algorithm (GAMSearch)* [6,7] which enables users to search for information within ConnectionLens graphs.

### 6.2 Integrating our DMV detection method within ConnectionLens

We now describe how we integrated our DMV detection approach described in Section 5 within ConnectionLens, to speed up graph construction. In a nutshell: we train our classifier during the graph construction process, then, in the same process, we use the classifier to predict which strings are DMVs, and thus, not worth the Named Entity Recognition effort.

**Value contexts** The DMV detection method applies to a set of values describing the same kind of data. Thus, while loading complex-structure data, we need to dynamically form such groups of values. For this, we view each value as occurring in a **context**, based on the structure of the dataset it comes from. On hierarchical data sources (JSON, XML, HTML, etc.), a natural context to attach to a value is the root-to-leaf label path on which the value occurs in a given document. In

PubMed bibliographic data, for instance, CoI values appear as text children of the XML elements on the path PubMedArticles.PubMedArticle.CoIStatement. Thus, we pair each value on which extraction could be performed (or which could be a DMV), with its context; all the values in a certain context form a set on which we can learn.

**Entity profiles (EP)** The next ingredient we need is *entity profiles* (Section 4), spelling which entities we expect to find in values from a certain context that are not DMVs; such a judgment is easily made by a human expert. Thus, currently, for each context where an entity profile is known, we allow users to specify them in a configuration file given to ConnectionLens.

**SmartExtract Web service** We deployed our DMV detection approach as a standalone Python Web service, called **smartExtract** in Figure 3, and also deployed using Flask. It is called by the Entity extraction module with each (value, context) pair, and works as follows (see Figure 4). To each context, we associate a state that represents the existence (or not) of a TF-IDF vectorizer and a trained Random Forest model (Section 5.2) able to predict, based on the entity profile (EP) associated with that context, whether a value occurring in the context is a DMV (Section 5). We consider the model has been sufficiently trained when it has reached a determined quality, measured by its f1-score. The quality to reach is a parameter of the service. For each value (text node) that ConnectionLens finds in that context (top box in Figure 4), we test this state (diamond box below the top box in the figure):

- *Yes*, the model associated with this EP has been fully trained on values previously encountered in the same context. In this case, we send the value to the smartExtract Web Service, which returns a boolean answer:
  - yes, this value is worth extracting entities from (through the Flair entity extraction service mentioned in Section 6.1); in this case, the value is added to the extraction batch;
  - no, this value is not worth the entity extraction effort, since our model predicts that the value is a DMV.
- *No*, the model for this EP has not been fully trained yet. In this case, our module needs to learn more about values in this context (by examining more results of the Flair extraction for values in this context). In this case, the node is added to the extraction Batch.

Once the extraction batch is full, ConnectionLens sends all the batch values to its Flair entity extraction service, to find out the entities contained in each value. We then capture these extraction results and share them with the smartExtract service to train its models (one different model for each entity profile). In turn, smartExtract may answer with the information that the state(s) associated with some context(s) have become true (the respective models are sufficiently trained). ConnectionLens keeps track of the context states on which it bases its decision (test box in Figure 4).

**Performance of online learning** In the above integration of smartExtract within ConnectionLens, the smartExtract models are continuously trained with

each value sent to extraction (*online learning*). In contrast, in the approach we described in Section 5, we trained the model once and for all with the training set, and then just used it to predict which values are DMVs. This difference has several consequences:

1. The training effort of our DMV technique drastically increases in this online integration within ConnectionLens.
2. To keep the TF-IDF representation of the data accurate with respect to all the data seen until a certain point during the graph construction, we need to compute a new vectorization and a new model for each new value.

While this online integration strategy achieves a high quality of data representation and prediction, it is computationally very expensive. Indeed, the cost of recomputing the vectorization and repeatedly re-training the model increases quickly with the number of nodes.

To reduce this high computational cost, we initially computed a new vectorization only when re-training the model (after each batch of values sent to the extractor). This strategy allowed us to save some time compared to the precedent, however, we were still losing time (when our goal is to save it) by using the smartExtract Web Service.

Next, we decided not to re-train the model after each batch, but to test it on each batch and re-train only if the quality of the model (as measured by the f1 prediction score) is under a determined threshold. This also brought some modest savings. However, using the smartExtract Web Service was still more computationally expensive than running the entity extractor on all values.

The change that actually brought visible performance benefits was to re-engineer the smartExtract service so that it does not load its model each time a prediction is made. Instead, we kept it in memory on the server side, which significantly reduced the time needed to call the smartExtract service and allowed it to speed up the ConnectionLens graph construction.

## 7 Experimental evaluation

In this section, we experimentally study several aspects related to the DMV methods that we have considered in the previous sections. After describing our datasets (Section 7.1) and experimental settings (Section 7.2), we show how FAHES performs on these datasets and that there is room for improvement in Section 7.3. Then, Sections 7.4 and 7.5 study DMV detection through entity profiling and through embedding (TF-IDF embedding and sentence-BERT embedding) and classification, respectively. In Section 7.6 we detail the results of our methods and of FAHES, over a small dataset manually labeled. Section 7.7 studies DMV detection impact in ConnectionLens extraction time.

### 7.1 Datasets

To conduct our experiments, we have built three ConnectionLens graphs out of real-world datasets. These were datasets Le Monde journalists suggested we

should integrate in ConnectionLens, for applications including that described in [6,8], and which inspired the research described in this paper. We believe these datasets are representative of many Open Data that is published through government and scientific transparency initiatives: data involving names of people, organizations, and partially obtained by asking individual to fill forms including free text fields.

1. We have loaded the most complete **HATVP XML** transparency dataset (35MB), with data about 270.000 people, in a ConnectionLens graph. From this, we extracted the *montant* (monetary amount) fields which appeared to contain many DMVs [7].
2. We loaded a smaller **HATVP CSV** dataset (2.1 MB), containing information about 9.000 people; this dataset is relational-looking, which simplifies processing it through FAHES.
3. We loaded 400.000 **PubMed bibliographic notices** in a graph, out of which we extracted (paper ID, conflict of interest statement) pairs. These CoI statements cover any kind of benefits (funding, personal fees, etc.) that authors report with various organizations such as companies, foundations, etc., as illustrated in Figure 1. PubMed data originates from various medical journals. Some do not provide CoI information; in this case, the CoI is an empty string. Others provide a default DMV, e.g., "The authors report no conflict". Finally, some journals only allow free text, leading to a large variety of DMVs.

We will denote these datasets as **DS1**, **DS2**, and **DS3**, respectively.

In practice, of course, we only extract entities once from each *distinct* string. It turns out that DS3 had a high number of duplicates (especially some very popular disguised nulls). Removing duplicates led to a new dataset we denote **DDS3** consisting of 82.388 values.

### 7.2 Settings

All experiments were performed on a MacBook Pro 16 inches from 2019, with a 2.4 GHz Intel Core i9 8-core processor and 32 GB 2667 MHz DDR4 memory. Experiments using sentence-BERT (Table 2) had to be run on Google Colab[8] because of the MacBook's lack of support for CUDA. For consistency, all results in Table 2 are obtained in Google Colab, with a Tesla-T4 GPU. We used ConnectionLens[9] to build the graphs, including in particular the extraction of named entities using Flair [5,4], which we had retrained for French [7]. ConnectionLens graphs are stored in Postgres 9.6; experiment code was written in Python 3.6.

**Precision, recall, and F1 measures** For our purposes, given that we aim to detect a certain form of DMV, a "positive" example is a DMV, while

[7] The transparency entry forms require filling in the worth of participations or ownerships in various companies; companies that have closed or did not make benefits, or only have a pro-bono activity, lead to DMVs.

[8] https://colab.research.google.com/

[9] Available from https://gitlab.inria.fr/cedar/connectionlens

**Table 1.** Entity extraction method times

| Values | Total characters | Extraction times (s) |
|---|---|---|
| 500 | 163.203 | 56 |
| 5.000 | 1.604.141 | 669 |
| 10.000 | 3.281.345 | 1.320 |
| 15.000 | 5.000.364 | 2.175 |
| 20.000 | 6.728.493 | 2.620 |

a "negative" example is an informative value. This interpretation is used in all the experiments described below. In the context of our project, *precision is more important than recall*, since low precision means wrongly considering a value as a DMV, while it contains valid information, which journalists would not want to miss. In comparison, low recall "only" means that we would waste time extracting entities from DMVs that turn out not to contain interesting information. While undesirable, the impact, from an application perspective, is lower.

### 7.3 DMV detection through FAHES

We have applied FAHES [25] on the three datasets described previously, asking it to detect DMVs. It is worth mentioning that FAHES is an unsupervised method; we use it for comparison as its goal of DMV detection is closest to our work. We comment on its results below.

**Results on DS1** Among the 270.000 values of the numeric *amount* attribute, FAHES correctly found the 0 (which occurs 45.000 times) as a Random DMV (Inlier). It also detected 372.2196 (4 occurrences) as a numerical outlier DMV; this is wrong. All the amount values are numbers, and as far as we could see, there are no other DMVs.

**Results on DS2** In this relational dataset, in an attribute called `filename`, FAHES identified **correctly** the DMV *dispense* (120 occurrences) as a Random DMV. Then, FAHES identified **wrongly** other values as being DMVs, in all cases as Syntactic Outliers DMV. The values falsely flagged as DMVs are:

- *François-Noël* (3 occurrences) in the attribute `given name`;
- *BÉRIT-DÉBAT* (6 occurrences) and *KÉCLARD-MONDÉSIR* (3) in the attribute `name`;
- *di* (4480 occurrences) in the attribute `document type`; this is the acronym for *déclaration d'intérêt*;
- *2A* and *2B* as departement numbers; they are, in fact, correct numbers of French departments in Corsica;
- four distinct, correct URLs within the `photo_url` attribute, probably because their structure did not resemble the others'.

DS2 seems to include no other DMVs.

**Results on DS3** Out of the 400.000 values, FAHES correctly identified *"The authors have declared that no competing interests exist"* (31.891 occurrences) as

a Random DMV. However, visual inspection exhibited many other DMVs (we will revisit this below). FAHES fails to find them because freely written texts rarely coincide, thus FAHES' statistical approach based on value frequencies leads it to consider a rarely occurring DMV as a valid value, which is wrong. All DMVs detected by FAHES on this dataset are actual DMVs (precision of 1.0). However, in this example, the entity profile technique identifies 351.123 DMVs, most of which FAHES misses, leading to a recall of $0,0909$. *This low recall shows the need for alternative techniques for detecting the DMVs we are interested in: uninformative, free-text answers.*

From these experiments, we conclude that FAHES fails to detect the DMVs we are interested in; DMVs detected as Syntactic outliers are often false positives. With a bit of domain knowledge, it is possible to manually discard these DMVs. However, importantly, FAHES has also shown its limitations on uninformative free text DMVs, by missing a vast majority of them. This shows the need of dedicated methods to detect such DMVs.

### 7.4 DMV detection through entity profiles

To measure performances of the entity profiles technique, we performed 5 experiments with respectively the 500, 5.000, 10.000, 15.000, and 20.000 first values of dataset the duplicate-free dataset DDS3. The objective here is to measure the extraction time as a function of the input size; this indicates the time needed to extract DMVs using the Entity Profile method. We present the results in Table 1. We observe that extracting entities is time consuming and that the extraction time is almost linear to the number of values. For the complete dataset DDS3, we can expect to have an extraction time of around 11.000 seconds (183 minutes), which is quite lengthy.

### 7.5 DMV detection through embedding and classification

The most common training-test split method consists on separating the dataset with 20% used for training and 80% for testing. We know that in our case, the most time-consuming operation is to label the training set with the entity extraction technique. Thus, to gain time, we want to reduce the training set.

To evaluate the impact of the training set size on the performance of the model used to detect DMVs over DDS3, we have performed three experiments. We trained models with respectively 20% (16.477 values), 10% (8.238 values), and 1% (823 values) of the dataset and report the comparison of the performances of each model in Table 2. In this table, the precision, recall, and thus F1 are computed using the result of the entity profile method (Section 4) as the gold standard. In bold, we highlight the best performance result between TF-IDF and sentence-BERT. Table 2 shows that we can attain very good precision, even if the model is trained on a small part of the dataset while saving significant amounts of time. Table 2 also shows that using sentence-BERT to represent our data does not significantly improve performance, while it heavily increases execution time. This happens despite speeding up its execution as much as possible,

**Table 2.** Impact of the training set size on the performance of DMV detection.

| Training-set size | 16.477 | | 8.238 | | 823 | |
|---|---|---|---|---|---|---|
| Embedding type | TF-IDF | s-BERT | TF-IDF | s-BERT | TF-IDF | s-BERT |
| Embedding Times (s) | 99 | 509 | 99 | 509 | 99 | 509 |
| Extraction Times (s) | 2.153 | 2.153 | 1.075 | 1.075 | 108 | 108 |
| Training Times (s) | 20 | 75 | 10 | 30 | 2 | 1 |
| Prediction Times (s) | 4 | 2 | 4 | 2 | 3 | 1 |
| Total Times (s) | **2.276** | 2.739 | **1.206** | 1.616 | **212** | 619 |
| Precision | 0,939 | **0,956** | 0,933 | **0,956** | 0,926 | **0,946** |
| Recall | **0,922** | 0,877 | **0,923** | 0,870 | **0,872** | 0,856 |
| F1-score | **0,931** | 0,915 | **0,928** | 0,911 | **0,899** | **0,899** |

**Table 3.** Comparing FAHES, Entity profile and classification over a sample of 200 manually-labeled values.

| Techniques | TP | FP | TN | FN | Precision | Recall | F1-score |
|---|---|---|---|---|---|---|---|
| FAHES | 0 | 0 | 96 | 104 | 0 | 0 | 0 |
| Entity profile | 82 | 4 | 92 | 22 | 0,953 | 0,788 | 0,862 |
| Classification TF-IDF | 94 | 2 | 94 | 10 | 0,979 | 0,904 | 0,940 |

by parallelizing execution over batches of 1.000. Interestingly, Table 2 also shows that in our problem, precision is less sensitive than recall to the reduction of the training-set size which suits our purpose.

With respect to our motivating example, building the graph for DDS3 took around 11.000 seconds. Using our method with a training-set of 1% of the values (823 values) takes now the time to predict to which values we have to apply the extractor (125 seconds), to which we add the time to extract the valuable values. We have found on our dataset that there are around 45.000 valuable values. We need 5.900 seconds to extract those. That brings us to a total of 6.000 **seconds to build our graph instead of** 11.000 **seconds previously**, without losing much information.

### 7.6 Comparison on manually labeled data

To get a better understanding of the performance of our entity profile technique, our classification technique, and FAHES, we labeled by hand a sample of 200 values from DDS3. The sample contains 96 actual CoIs, and 104 DMVs, labeled following the instructions of our domain expert (journalist).

Table 3 shows, for each method, the numbers of true and false negatives (TN and FN), as well as the precision, recall, and F1 score of each technique *with respect to the human-labeled gold standard*. Our first observation is that FAHES has not found a single DMV in this dataset, leading to recall, precision, and F1 of 0.

The Entity Profile technique performed quite well (F1 of $0,86$). A few valid CoI statements are wrongly detected as DMVs, but the contrary is quite rare, and that suits our application needs since we should not skip extraction on valid CoIs.

Our classification method performed even better (F1 of $0,94$). Since this method is based on word frequency, we can suppose that some words are specific to DMVs and others are specific to valid CoIs.

Inspecting the false negatives (DMVs detected as valid CoIs), we noticed that they mention "ICMJE", as in *Conflicts of Interest: All authors have completed the ICMJE uniform disclosure form (available at http://dx.doi.org/10.21037/atm-20-3650). The authors have no conflicts of interest to declare.* ICMJE has been wrongly detected as an entity, which led astray our entity profile technique. As our classification model is trained with entity profile results, "ICMJE" has probably been identified as associated with valid CoIs.

On the other hand, the entity profile technique has led to only having 4 CoIs identified as DMVs (false positives). In those, the authors mentioned patents they have, which may be seen as creating a conflict of interest with the research presented in the paper. However, by the domain experts' (journalists') rule, these statements are no conflicts of interest.

Finally, we found that the NER has missed an organization entity in the value *"The authors are supported by the use of resources and facilities at the Michael E. Debakey VA Medical Center, Houston, Texas"*, thus the entity profile detection technique has wrongly considered this string to be a DMV. However, the classification model did not make the same mistake and correctly considered this value an informative COI.

### 7.7 DMV detection integrated in ConnectionLens

As explained in Section 6.2, we developed a Python Flask service for DMV detection based in entity profiles, to save extraction time in specified contexts. We only kept TF-IDF representation, because sentence-BERT has proven to be way slower. To measure its impact on the time spent extracting entities, we loaded 100.000 XML PubMed bibliographic notices into ConnectionLens, without the smartExtract service (that is, following the graph construction method prior to this work), then also using the smartExtractor service as described in Section 6.2. The quality parameter (F1-score mentioned in Section 6.2) is set to 0.9. To mesure the F1-score, we split each batch of values used for training, into training testing subsets, the latter containing 20% randomly chosen values from the batch. We used a batch size of 160, that is: the Flair NER entity extractor was called on groups of 160 values. In this experiment, the smartExtract service is called to ConflictOfInterest values, for which an entity profile is manually provided. Thus, in this experiment, we only apply entity extraction on the ConflictOfInterest values.

As we can see in Table 4, we saved 557 calls to the Flair NER Service (which amounts to 70% of the calls) thanks to the smartExtract service. This corresponds to saving 1.770 seconds on extraction (33% of the extraction time) and 2.215 seconds from the total graph construction time. Finally, using our smartExtract service leads to missing out on 3.225 organization entities that would have been extracted without it. These represent about 10% of the total orga-

**Table 4.** Impact of the smartExtract web service on entity extraction within ConnectionLens.

| Is the smartExtract web service used? | Yes | No |
|---|---|---|
| Number of calls to Flair NER service | 228 | 785 |
| Time needed for extraction (seconds) | 3.514 | 5.284 |
| Total graph construction time (seconds) | 6.865 | 9.080 |
| Number of organization entities extracted | 35.110 | 38.335 |

nization entities, which is consistent with our quality parameter (F1-score = 0.9).

The overall time saving is slightly higher than the one corresponding only to extraction. This is because not creating some entities also means we do not have to store them, nor the edges that connect them to their parents in the graph.

## 8 Conclusion

Integration of very heterogeneous data, such as we encountered in data journalism applications [8,9], requires extracting Named Entities from all values found in the data, in order to interconnect sources through the presence of the same entity in two or more sources. The ConnectionLens [7,6] system we have developed leverages this idea.

In this work, we sought to automatically find *uninformative answers*: these are free texts that users write in response to some question, and which do not provide the information required by the question. These can be seen as a particular class of textual DMVs. While many kinds of DMVs have been studied in the literature, as we discussed in Section 2 and 5.3, when uninformative answers are basically all distinct, existing methods cannot detect them.

The first technique we studied leverages ConnectionLens' effort carried to extract Named Entities from each value of the database. Our technique exploits so-called *entity profiles* (expected entities in a valid value) to identify DMVs. While highly accurate, this is expensive time-wise, because of the extraction. For efficiency, instead, our second method trained a Random Forest classifier on a small subset of our dataset, labeled with entity profiles, and classified the other values as DMVs or not. This technique saves significant extraction time, while also having very good accuracy.

We have included this method within the ConnectionLens platform and demonstrated that it allows to avoid a large part of the Named Entity Recognition (NER) effort. This is significant, since as shown in [7], NER dominates the graph construction time. Thus, the method proposed here allows speeding up the construction of integrated graphs out of heterogeneous data sources.

**Acknowledgments** This work was supported by the ANR AI Chair project SourcesSay Grant no ANR-20-CHIA-0015-01. Galhardas' work was supported by national funds through FCT under the project UIDB/50021/2020.

## References


1. Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. Detecting data errors: Where are we and what needs to be done? *Proc. VLDB Endow.*, 9(12):993–1004, aug 2016.
2. Ziawasch Abedjan, Lukasz Golab, Felix Naumann, and Thorsten Papenbrock. *Data Profiling*. Morgan and Claypool, 2020.
3. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
4. Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. Flair: An easy-to-use framework for state-of-the-art NLP. In *ACL*, 2019.
5. Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *ACL*, 2018.
6. Angelos-Christos Anadiotis, Oana Balalau, Théo Bouganim, Francesco Chimienti, Helena Galhardas, Mhd Yamen Haddad, Stéphane Horel, Ioana Manolescu, and Youssr Youssef. Empowering Investigative Journalism with Graph-based Heterogeneous Data Management. *Bulletin of the Technical Committee on Data Engineering*, September 2021.
7. Angelos Christos Anadiotis, Oana Balalau, Catarina Conceicao, Helena Galhardas, Mhd Yamen Haddad, Ioana Manolescu, Tayeb Merabti, and Jingmao You. Graph integration of structured, semistructured and unstructured data for data journalism. *Information Systems*, 2021.
8. Angelos-Christos G. Anadiotis, Oana Balalau, Theo Bouganim, Francesco Chimienti, Helena Galhardas, Mhd Yamen Haddad, Stephane Horel, Ioana Manolescu, and Youssr Youssef. Discovering conflicts of interest across heterogeneous data sources with connectionlens. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 4670–4674. ACM, 2021.
9. Raphaël Bonaque, Tien Duc Cao, Bogdan Cautis, François Goasdoué, Javier Letelier, Ioana Manolescu, Oscar Mendoza, Swen Ribeiro, Xavier Tannier, and Michaël Thomazo. Mixed-instance querying: a lightweight integration architecture for data journalism. In *VLDB*, 2016.
10. Théo Bouganim, Helena Galhardas, and Ioana Manolescu. Efficiently identifying disguised nulls in heterogeneous text data. In *BDA (Conférence sur la Gestion de Données – Principles, Technologies et Applications)*, Paris, France, October 2021. Informal publication only.
11. Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
12. Camille Chanial, Rédouane Dziri, Helena Galhardas, Julien Leblay, Minh-Huong Le Nguyen, and Ioana Manolescu. ConnectionLens: Finding connections across heterogeneous data sources (demonstration). *PVLDB (also at BDA)*, 11(12), 2018.
13. Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13, June 1970.
14. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *ACL*, 2019.
15. Helena Galhardas, Daniela Florescu, Dennis E. Shasha, and Eric Simon. Declaratively cleaning your data with AJAX. In Anne Doucet, editor, *BDA*, 2000.

16. Helena Galhardas, Daniela Florescu, Dennis E. Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, 2001.
17. Joachim Hammer, Hector Garcia-Molina, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. In *SIGMOD*, 1995.
18. Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining Concepts and Techniques*. Morgan Kaufmann, 2011.
19. Ming Hua and Jian Pei. Cleaning disguised missing data: A heuristic approach. In *SIGKDD*, 2007.
20. Ihab F. Ilyas and Mohamed A. Soliman. *Probabilistic Ranking Techniques in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
21. Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 1987. First edition.
22. Mohammad Mahdavi and Ziawasch Abedjan. Baran: Effective error correction via a unified context representation and transfer learning. *Proc. VLDB Endow.*, 13(12):1948–1961, jul 2020.
23. Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 865–882, New York, NY, USA, 2019. Association for Computing Machinery.
24. Ronald K. Pearson. The problem of disguised missing data. *SIGKDD Explor. Newsl.*, 8(1):83–92, June 2006.
25. Abdulhakim A. Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. Fahes: A robust disguised missing values detector. In *SIGKDD*, 2018.
26. Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, 2001.
27. Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *EMNLP*, 2019.