Extracting Linked Data from statistic spreadsheets

Tien-Duc Cao, Ioana Manolescu, Xavier Tannier

Extracting Linked Data from statistic spreadsheets

Tien Duc Cao
Inria and LIX
(CNRS & Ecole Polytechnique)
Université Paris-Saclay
tien-duc.cao@inria.fr

Ioana Manolescu
Inria and LIX
(CNRS/Ecole Polytechnique)
Université Paris-Saclay
ioana.manolescu@inria.fr

Xavier Tannier
Sorbonne Université
Inserm
LIMICS
xavier.tannier@sorbonne-universite.fr

ABSTRACT

Fact-checking journalists typically check the accuracy of a claim against some *trusted* data source. *Statistic databases* such as those compiled by state agencies are often used as trusted data sources, as they contain valuable, high-quality information. However, their usability is limited when they are shared in a format such as HTML or spreadsheets: this makes it hard to find the most relevant dataset for checking a specific claim, or to quickly extract from a dataset the best answer to a given query.

In this work, we provide a conceptual model for the open data comprised in statistics published by INSEE, the national French economic and societal statistics institute. Then, we describe a novel method for extracting RDF Linked Open Data, to populate an instance of this model. We used our method to produce RDF data out of 20k+ Excel spreadsheets, and our validation indicates a 91% rate of successful extraction.

Further, we also present a novel algorithm enabling the exploitation of such statistic tables, by (*i*) identifying the statistic datasets most relevant for a given fact-checking query, and (*ii*) extracting from each dataset the best specific (precise) query answer it may contain. We have implemented our approach and experimented on the complete corpus of statistics obtained from INSEE, the French national statistic institute.

Our experiments and comparisons demonstrate the effectiveness of our proposed method.

1 INTRODUCTION

The tremendous value of Big Data has been noticed of late also by the media, and the term "data journalism" has been coined to refer to journalistic work inspired by data sources [14]. While data of some form is a natural ingredient of all reporting, the increasing volumes of available digital data as well as its increasing complexity lead to a qualitative jump, where technical skills for working with data are stringently needed in journalism teams.

A class of data sources of particularly high value is that comprised into *government statistics*. Such data has been painstakingly collected by state administrations which sometimes have very significant human and technology means at their disposal. This makes the data oftentimes of high quality. Another important quality of such data is that it logically falls into the open data category, since it is paid for with taxpayer money. Example of such government open data portals http://data.gov (US), http://data.gov.uk (UK), http://data.gouv.fr and http://insee.fr (France), http://wsl.iiitb.ac.in/sandesh-web/sandesh/index (India) etc.

Such high-quality statistic data is a very good background information to be used when trying to assess the truthfulness of a claim. A fact-checking scenario typically has several steps: first, a *claim* is made. Second, journalists or other fact-checkers look up *reference sources* providing relevant statistics for the assessment of the claim; Third, some *post-processing* may be applied to compute from the reference source values, the value corresponding to the claim. This post-processing may involve elaborate steps.

Ideally, these are available to fact-checkers in a format and organized according to a model that they understand well. For instance, in [31], unemployment data is assumed available in a temporal database relation. In reality, however, most fact-checking actors only have access to the data through *publication* by the institute. This raises the question of the format and organization of the data. While the W3C's best practices argue for the usage of RDF in publishing open data [27], in practice open data may be published Excel, HTML or PDF; in the latter formats, statistic data is typically shown as tables or charts. In the particular case of the **INSEE** (http://www.insee.fr), the national French social and economic statistics institute, while searching for some hot topics in the current French political debate (youth unemployment, immigrant population etc.) we found many very useful datasets in Excel and/or HTML, sometimes accompanied by a PDF report explaining and commenting the data, but we did not find the data in a structured, machine-readable format.

Motivated by this limitation, we propose an approach for extracting Linked Open Data from INSEE Excel tables. The two main contributions of our work are: (*i*) a conceptual data model (and concrete RDF implementation) for statistic data such as published by INSEE and other similar institutions, and (*ii*) an extraction algorithm which populates this model based on about 11,000 pages including 20,743 Excel files published by INSEE. We applied our approach to the complete set of statistics published by INSEE, a leading French national statistics institute, and republish the resulting RDF Open Data (together with the crawling code and the extraction code[1]). While our approach is tailored to some extent toward French, its core elements are easily applicable to another setting, in particular for statistic databases using English, as the latter is very well-supported by text processing tools.

Subsequently, we develop a system to *search for answers to keyword queries in a database of statistics*, organized in RDF graphs such as those we produced. Thus, the focus of this system is: *given a claim, identify from a corpus of statistics made available as tabular (Excel) files, the information most likely to be useful as background against which to fact-check the claim*. Given the variety of data sources and the natural-language nature of a claim, we view this as a top-k search problem, and aim at presenting to the user *scored results*, in the decreasing order of their matching the claim. We have worked in particular on INSEE data sources, however from our observations, a similar approach can be mapped out to exploit other organizations'

[1] https://gitlab.inria.fr/cedar/insee-crawler, https://gitlab.inria.fr/cedar/excel-extractor

statistic databases, such as the ones available at http://www.data.gov or https://data.gov.uk. First, we describe a *dataset search* algorithm, which given a set of user keywords, identifies the datasets (statistic table and surrounding presentations) most likely to be useful for answering the query. Second, we devised an *answer search* algorithm which, building on the above algorithm, attempts to answer queries, such as "unemployment rate in Paris in 2016", with *values* extracted from the statistics dataset, together with a *contextualization* quickly enabling the user to visually check the correctness of the extraction and the result relevance. In some cases, there is no single number known in the database, but several semantically close ones. In such cases, our algorithm returns the set of numbers, again together with context enabling its interpretation.

We have experimentally evaluated the efficiency and the effectiveness of our algorithms, and demonstrate their practical interest to facilitate fact-checking work. In particular, while political debates are broadcast live on radio or TV, fact-checkers can use such tools to quickly locate reference numbers which may help them publish live fact-checking material.

Below, we describe the data sources we work with and the model for the extracted data (Section 2) and our extraction method (Section 3). We define the search problem that we address in Section 4 and then describe our search algorithms. In Section 5 we present our experimental evaluation. We discuss related work in Section 6, then conclude.

2 REFERENCE STATISTIC DATA

INSEE data sources INSEE publishes a variety of datasets in many different formats. In particular, we performed a crawl of the INSEE web site under the categories *Data/Key figures*, *Data/Detailed figures*, *Data/Databases*, and *Publications/Wide-audience publications*, which has lead to about 11,000 web pages including 20,743 Excel files (with total size of 36GB) and 20,116 HTML tables. In this work, we focused on the set of Excel files; we believe our techniques could be quite easily adapted to HTML tables in the future. Also, while we targeted INSEE in this work, we found similar-structure files published in other Open Data government servers in the UK (e.g., http://www.ic.nhs.uk/catalogue/PUB08694/ifs-uk-2010-chap1-tab.xls) and the US (e.g., http://www.eia.gov/totalenergy/data/monthly/query/mer_data_excel.asp?table=T02.01).

We view each spreadsheet file as a collection of *data sheets* $D_1, D_2, \ldots$ Each data sheet D has a *title* $D.t$ and optionally an *outline* $D.o$. The title and outline are given by the statisticians producing the spreadsheets in order to facilitate their understanding by human users, such as the media and general public. The title is a short nominal phrase stating what D contains, e.g., "Average mothers' age at the birth of their child per department in 2015". The outline, when present, is a short paragraph which may be found on a Web page from where the file enclosing D can be downloaded. The outline extends the title, for instance to state that the mothers' ages are rounded to the closest smaller integers, that only the births of live (not stillborn) children are accounted for etc.

In a majority of cases (about two thirds in our estimation), each data sheet D comprises *one* statistics table, which is typically a *two-dimensional aggregate table*. Data is *aggregated* since statistics institutes such as INSEE are concerned with building such global measures of society or economy phenomena, and the aggregate tends to be *bidimensional* since they are meant to be laid out in a format easy for humans to read. For instance, Figure 1 illustrates a data sheet for the birth statistic dataset mentioned above. In this example, the two dimensions are the mother's age interval, e.g., "16-20", "21-25", "26-30" etc. and her geographic area, specified as a department, e.g., "Essonne", "Val-de-Marne" etc. For a given age interval and region, the table includes the number of women which were first-time mothers in France in 2015, in that age interval and department. In Figure 1, we have included a top row and column on a gray background, in order to refer to the row and column numbers of the cells in the sheet. We will use the shorthand $D_{r,c}$ to denote the cell at row r and column c in D. We see that the table contains *data cells*, such as $D_{6,3}$, $D_{6,4}$, which hold data values, etc. and *header cells*, e.g., "Region", "Age below 30" etc., which (*i*) characterize (provide context for) the data values, and (*ii*) may occupy several cells in the spreadsheet, as is the case of the latter.

The basic kind of two-dimensional agregate in a data sheet is (close to) the result of a group-by query, as illustrated in Figure 1. However, we also found cases where the data sheet contains a *data cube* [15], that is: the result of a group-by, enhanced with partial sums along some of the group-by dimensions. For instance, in the above example, the table may contain a row for every region labeled "Region total" which sums up the numbers for all departments of the region, for each age interval (e.g., https://www.insee.fr/fr/statistiques/fichier/2383936/ARA_CD_2016_action_sociale_1-Population_selon_l_age.xls).

Dimension hierarchies are often present in the data. For instance, the dimension values "16-20", "21-25", "26-30" are shown in Figure 1 as subdivisions of "Age below 30", while the next four intervals may be presented as part of "Age above 31". The age dimension hierarchy is shown in magenta in Figure 1, while the geographical dimension hierarchy is shown in blue. In those cases, the lower-granularity dimension values appear in the table close *close to the finer-granularity dimension values which they aggregate*, as exemplified in Figure 1. *Cell fusion* is used in this case to give a visual cue of the finer-granularity dimension values corresponding to the lower-granularity dimension value which aggregates them.

A few more rules govern data organization in the spreadsheet:
(1) $D_{1,1}$ (and in some cases, more cells on row 1, or the first few rows of D) contain the outline $D.o$, as illustrated in Figure 1; here, the outline is a long text, thus a spreadsheet editor would show it over several cells. One or several fully empty rows separate the outline from the topmost cells of the data table; in our example, this is the case of row 2. Observe that due to the way data is spatially laid out in the sheet, these topmost cells are related to the lowest granularity (top of the hierarchy) of one dimension of the aggregate. For instance, in Figure 1, this concerns the cell at line 3 and columns 3 to 9, containing **Mother's age at the time of the birth**.
(2) Sometimes, the content of a header cell (that is, a value of an aggregation dimension) is not human-understandable, but it is the name of a variable or measure, consistently used in all INSEE publications to refer to a given concept. Exactly in these cases, the file containing D has a sheet immediately after D, where the variable (or measure) name is mapped to the natural-language description explaining its meaning. For example, we translate SEXE1_AGEPYR1018

l \ c	1	2	3	4	5	6	7	8	9	10
1	The data reflects children born alive in 2015...									
2										
3			Mother's age at the time of the birth							
4			Age below 30			Age above 31				
5	**Region**	**Department**	**16-20**	**21-25**	**26-30**	**31-35**	**36-40**	**41-45**	**46-50**	
6	Île-de-France	Essonne	215	1230	5643	4320	3120	1514	673	
7		Val-de-Marne	175	987	4325	3156	2989	1740	566	
8		...	...	...	...	...	...	...	...	
9	Rhône-Alpes	Ain	76	1103	3677	2897	1976	1464		
10		Ardèche	45	954	2865	2761	1752	1653	523	
11		...	...	...	...	...	...	...	...	
...	...	...	...	...	...	...	...	...	...	

Figure 1: Outline of a sample statistics table in an INSEE dataset.

(https://www.insee.fr/fr/statistiques/fichier/2045005/BTX_TD_POP1A_2013.zip) into *Male from 18 to 24 years old* using the table below:

Variable	Meaning
SEXE	**Sex**
1	Male
2	Female
AGEPYR10	**Age group**
00	Less than 3 years old
03	3 to 5 years old
06	6 to 10 years old
11	11 to 17 years old
...	...
80	80 years and older

Conceptual data model From the above description of the INSEE statistic datasets, we extract a conceptual data model shown in Figure 2. Entities are shown as boxes while attributes apper in oval shapes. We simplified the notation to depict relationships as directed arrows, labeled with the relationship name. *Data cells* and *header cells* each have an attribute indicating their value (content); by definition, any data or header cell has a non-empty content. Moreover, a data cell has one associated *location*, i.e., (row number, column number), while a header cell may occupy one or several (adjacent) locations. Each data cell has a closest header cell on its column, and another one on its line; these capture the dimension values corresponding to a data cell. For instance, the closest column header cell for the data cell at $D_{7,4}$ is $D_{5,4}$, while the closest column header cell is $D_{7,2}$. Further, the header cells are organized in an aggregation hierarchy materialized by the respective relation "aggregated by". For instance, $D_{5,4}$ is aggregated by $D_{4,3}$, which is agregated by $D_{3,3}$. In each sheet, we identify the top dimension values as those which are not aggregated by any other values: in our example, the top dimension values appear in $D_{3,3}$, $D_{6,1}$ and $D_{9,1}$. Note that *such top dimension values are sometimes dimensions names*, e.g., **Region**, and other times *dimensions values*, e.g., Île-de-France. *Statistics in a sheet are not always organized exactly as predicted by the relational aggregate query models*, even if they are often close.

Finally, note that our current data model does not account for data cubes, that is: we do not model the cases when some data cells are

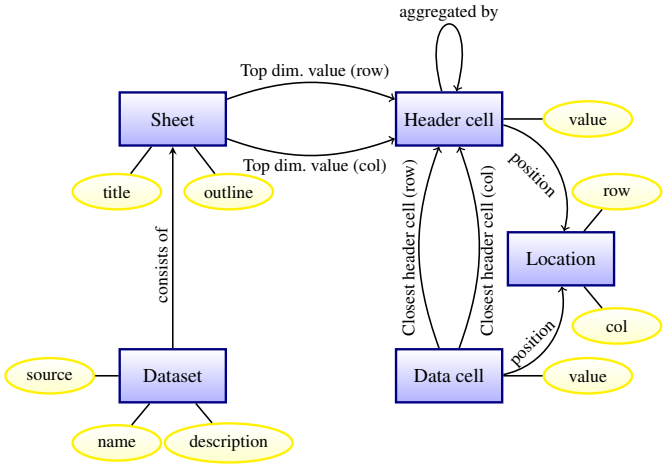

Figure 2: Conceptual data model.

aggregations over the values of other data cells. Thus, our current extraction algorithm (described below) would extract data from such cubes as if they were result of a plain aggregation query. This clearly introduces some loss of information; we are working to improve it in a future version.

3 EXTRACTION

We developed a tool for extracting data from spreadsheet files so as to populate an instance of our conceptual data model; it uses the Python library xlrd (https://pypi.python.org/pypi/xlrd) for accessing spreadsheet content. Below, we present the extraction procedure. Unlike related work (see Section 6), it is not based on a sequence classification with a supervised learning approach, but on a more global rule-based system aiming at framing the data cell range and extracting the headers around these data cell.

For each sheet, the first task is to *identify the data cells*. In our corpus, these cells contain *numeric* information, that is: integers or real numbers, but also interval specifications such as "20+" (standing for "at least 20"), special *null* tokens such as "n.s." for "not specified".

Our approach is to identify the *rectangular area containing the main table (of data cells)*, then build data cells out of the (row, column) locations found in this area. Note that the area may also contain empty locations, corresponding to the real-life situation when an information is not available or for some reason not reported, as is the case for $D_{9,9}$ in Figure 1.

Data cell identification proceeds in several steps.

DCI1 Identify the leftmost data location (ldl) on each row: this is the first location encountered from left to right which (i) follows at least one location containing text (such locations will be interpreted as being part of header cells), and (ii) has a non-empty content that is compatible with a data cell (as specified above). We store the column numbers of all such leftmost data locations in an array called ldl; for instance, $ldl[1] = -1$ (no data cell detected on row 1) and the same holds for rows 2 to 5, while $ldl[i] = 3$ for $i = 6, 7$ etc.

DCI2 For each row i where $ldl[i] \neq -1$, we compute a *row signature* $sig[i]$ which is counts the non-empty locations of various types found on the row. For instance, $sig[6] = \{string : 2, num : 7\}$ because the row has 2 string-valued cells ("Île-de-France" and "Essonne") and 7 numerical cells. For each signature thus obtained, we count the number of rows (with data cells) having exactly that signature; the largest row set is termed the *data table seed*. Note that the rows in the data table seed are not necessarily contiguous. This reflects the observation that slightly different-structured rows are sometimes found within the data table; for instance, $D_{9,9}$ is empty, therefore row 9 is not part of the seed, which in this example, consists of the rows {6,7,8,10,11} and possibly other rows (not shown) below.

DCI3 We "grow" the data table seed it by adding *every row at distance 1 or at most 2 above and below the data table seed, and which contains at least a data cell*. We do this repeatedly until no such rows can be found. In our example, this process adds row 9, despite the fact that it does not have the same signature as the others. We repeat this step in an inflationary fashion until no row adjacent to the current data table qualifies. The goal of this stage is to make sure we do not miss interesting data cells by being overly strict w.r.t. the structure of the rows containing data cells.

At this point, all the data cells are known, and their values and coordinates can be extracted.

Identification and extraction of header cells We first look for those organized in *rows*, e.g., rows 3, 4 and 5 in our example.

HCE1 We attempt to exploit the visual layout of the file, by looking for a horizontal *border* which separates the first header row of the (usually empty) row above it.

HCE1.1 When such a border exists (which represents a majority but not all the cases), it is a good indicator that header cells are to be found *in the rows between the border and going down until (and before) the first data row*. (Note that the data rows are already known by now.) In our example, this corresponds to the rectangular area whose corners are $D_{3,1}$ and $D_{4,9}$.

HCE1.2 When the border does not exist, we inspect the rows starting immediately above the first data row and moving up, as follows. In each row i, we examine the locations starting at the column $ldl[i]$ (if this is greater than 0). If the row doesn't contain empty locations, we consider that it is part of the header area. Intuitively, this is because no data cell can lack a value along one of the dimensions, and header cells carry exactly dimension values. An empty header cell would correspond precisely to such a missing dimension value.

HCE2 Once the header area is known, we look for conceptual header cells. The task is complicated by the fact that such a cell may be spread over several locations: typically, a short text is spread from the top to the bottom over locations situated one on top of the other. This can also happen between locations in the same row (not in the same column), which are fused: this is the case of $D_{3,3}$ to $D_{3,9}$ in our example. The locations holding a single dimension value are sometimes (but not always) fused in the spreadsheet; sometimes, empty locations are enclosed within the same border as some locations holding text, to obtain a visually pleasing aspect where the dimension value "stands out well". We detect such groups by *tracking visible cell borders*, and create one conceptual header cell out of each area border-enclosed area.

HCE3 Next, we *identify and extract header cells organized in columns*, e.g., columns 1 and 2 in our example. This is done by traversing all the data rows in the data area, and collecting from each row i, the cells that are at the left of the leftmost data cell $ldl[i]$.

Finally, we populate the instance of the data model with the remaining relationships we need. We connect: each data and header cell to its respective locations; each data cell with its closest header cell on the same column, and with its closest header cell on the same row; each row (respectively, column) header cell to its closest header cell above (respectively, at left) through the relation aggregated by.

4 SEARCH PROBLEM AND ALGORITHM

Given a *keyword-based query*, we focus on returning a *ranked list of candidate answers*, ordered in the decreasing likelihood that they contain (or can be used to compute) the query result. A relevant candidate answer can be a data cell, a data row, column or even an entire dataset. For example, consider the query "youth unemployment in France in August 2017". An Eurostat dataset[2] is a good candidate answer to this query, since, as shown in Figure 3, it contains one data cell, at the intersection of the France row with the Aug-2017 column. Now, if one changes the query to ask for "youth unemployment in France in 2017", no single data cell can be returned; instead, all the cells on the France row qualify. Finally, a dataset containing 2017 French unemployment statistics over the general population (not just youth) meets some of the search criteria (2017, France, unemployment) and thus may deserve to appear in the ranked list of results, depending on the availability of better results.

This task requires the development of specific novel methods, borrowing ideas from traditional IR, but following a new methodology. This is because our task is very specific: we are searching for information not within text, but within tables, which moreover are not flat, first normal form database relations (for which many keyword search algorithms have been proposed since [16]), but partially nested tables, in particular due to the hierarchical nature of the headers, as we explained previously. While most of the reasoning performed by our algorithm follows the two-dimensional layout of data in columns and tables, bringing the data in RDF: (i) puts a set of interesting, high-value data sources within reach of developers

[2] http://ec.europa.eu/eurostat/statistics-explained/images/8/82/Extra_tables_Statistics_explained-30-11-2017.xlsx

and (ii) allows us to query across nested headers using regular path queries expressed in SPARQL (as we explain in Section 4.6).

We describe our algorithms for finding such answers below.

4.1 Input data and architecture

System architecture A scraper collects the Web pages publicly accessible from a statistic institute Web site; Excel and HTML tables are identified, traversed, and converted into RDF graphs as explained above. The resulting RDF data is stored in the Apache Jena TDB server[3], and used to answer keyword queries as we explain next.

	Seasonally adjusted youth (under 25s) unemployment				
	Number of persons (in thousands)				
	Oct-2016	Jul-2017	Aug-2017	Sep-2017	Oct-2017
Belgium	:	77	77	77	:
Bulgaria	27	22	21	19	19
Czech Republic	34	27	25	23	23
Denmark	62	54	53	49	47
Germany	293	283	283	283	283
France	663	629	625	623	625

Figure 3: Sample dataset on French youth unemployment.

RDF data extracted from the INSEE statistics INSEE publishes a set $\mathcal{D}$ of datasets $D_1, D_2, \ldots$ etc. where each dataset D_i consists of a table (HTML table or a spreadsheet file), a *title* $D_i.t$ and optionally an *outline* $D_i.o$. The title is a short nominal phrase stating what D_i contains, e.g., "Seasonally adjusted youth (under 25s) unemployment" in Figure 3. The outline, when present, is a short paragraph on the Web page from where D_i can be downloaded. The outline extends the title with more details about the nature of the statistic numbers, the interpretation of each dimension, methodological details of how numbers were aggregated etc.

A *dataset* consists of *header cells* and *data cells*. Data cells and header cells each have attributes indicating their position (column and row) and their value. For instance, in Figure 3, "Belgium", "Bulgaria" etc. up to "France", "Oct-2016" to "Oct-2017", and the two fused cells above them, are header cells, while the other are data cells. Each data cell has a *closest header cell* on its column, and another one on its row; these capture the dimension values corresponding to a data cell.

To enable linking the results with existing (RDF) datasets and ontologies, we extract the table contents into an RDF graph. We create *an RDF class for each entity type*, and assign *an URI to each entity instance*, e.g., dataset, table, cell, outline etc. Each relationship is similarly represented by a resource described by the entity instances participating to the relationship, together with their respective roles.

4.2 Dataset search

The first problem we consider is: given a keyword query Q consisting of a set of keywords u_1, u_2, …, u_m and an integer k, find the k datasets most relevant for the query (we explain how we evaluate this relevance below).

We view each dataset as a *table* containing a *title*, possibly a *comment*, a set of *header cells* (row header cells and column header cells) and a set of *data cells*, the latter containing numeric data[4]. At query time, we transform the query Q into a set of of keywords $W = w_1, w_2, \ldots, w_n$ using the method described in Section 4.3. Offline, this method is also used to transform each dataset's text to words and we compute the score of each word with respect to a dataset, as described in Section 4.4. Then, based on the word-dataset score and W, we estimate datasets' relevance to the query as we explain in Section 4.5.

4.3 Text processing

Given a text t (appearing in a title, comment, or header cell of the dataset, or the text consisting of the set of words in the query Q), we convert it into a set of words using the following process:

- First, t is *tokenized* (separated into distinct words) using the KEA[5] tokenizers. Subsequently, each multi-word French location found in t that is listed in Geonames[6], is put together in a single token.
- Each token (word) is converted to lowercase, stop words are removed, as well as French accents which complicate matching.
- Each word is mapped into a word2vec vector [23], using the gensim [24] tool. Bigrams and trigrams are considered following [23]. We had trained the word2vec model on a general-domain French news web page corpus.

4.4 Word-dataset score

For each dataset extracted from the statistic database, we compute a score characterizing its semantic content. A first observation is that datasets should be returned not only when they contain the exact words present in the query, but also if they contain very closely related ones. For example, a dataset titled "Duration of marriage" could be a good match for the query "Average time between marriage and divorce" because of the similarity between "duration" and "time". To this effect, we rely on *word2vec* [23] which provides similar words for any word in its vocabulary: if a word w appears in a dataset D, and w is similar to w', we consider w' also appears in D.

The score $score(w)$ of a dataset D w.r.t the query word w is 1 if w appears in D.

If w does not appear in D:

- If there exists a word w', from the list of top-50 similar words of w according to word2vec, which appears in D, then $score(w)$ is the similarity between w and w'. If there are several such w', we consider the one most similar to w.
- If w is the name of a Geonames place we can't apply the above scoring approach because "comparable" places (e.g., cities such as Paris and London) will have high similarity in the word2vec space. As the result, when user asks for "unemployment rate Paris", the data of London might be returned instead of Paris's. Let p be the number of places that Geonames' hierarchy API[7] returns for w (p is determined

[3] https://jena.apache.org/documentation/tdb/index.html

[4] This assumption is backed by an overwhelming majority of cases given the nature of statistic data. We did encounter some counterexamples, e.g. http://ec.europa.eu/eurostat/cache/metadata/Annexes/mips_sa_esms_an1.xls. However, these are very few and thus we do not take them into account in our approach.

[5] https://github.com/boudinfl/kea

[6] http://www.geonames.org/

[7] http://www.geonames.org/export/place-hierarchy.html

by Geonames and depends on w). For instance, when querying the API with *Paris*, we obtain the list *Île-de-France, France, Europe*. Let w'_i be the place at position i, $1 \leq i \leq p$ in this list of returned places, such that w'_i appears in D. Then, we assign to D a score for w equal to $(p+1-i)/(p+1)$, that is, the most similar place according to Geonames has rank $p/(p+1)$, and the least similar has the rank $1/(p+1)$. If D contains several of the places from the w's hierarchy, we assign to D a score for w corresponding to the highest-ranked such place.

- 0 otherwise.

Based on the notion of word similarity defined above, we will write $\boxed{w \prec W}$ to denote that the word w from dataset D either belongs to the query set W, or is close to a word in W. Observe that, by definition, for any $w \prec W$, we have $score(w) > 0$.

We also keep track of the *location(s)* (title, header and/or comment) in which a word appears in a dataset; this information will be used when answering queries, as described in Section 4.5. In summary, for each dataset D and word $w \in D$ such that $w \prec W$, we compute and store tuples of the form:

$$(w, score(w), location(w, D), D)$$

where $location(w, D) \in \{\mathsf{T}, \mathsf{HR}, \mathsf{HC}, \mathsf{C}\}$ indicates where w appears in D: T denotes the title, HR denotes a row header cell, HC denotes a column header cell, and C denotes an occurrence in a comment.

These tuples are encoded in JSON and stored in a set of files; each file contains the scores for *one* word (or bigram) w, and *all* the datasets.

4.5 Relevance score function

Content-based relevance score function. This function, denoted $g_1(D,W)$, quantifies the interest of dataset D for the word set W; it is computed based on the tuples $(w, score(w), location(w, D), D)$ where $w \prec W$.

We experimented with many score functions that give high ranking to datasets that have many matching keywords (see details in section 5.2.2). These functions are monotonous in the score of D with respect to each individual word w. This enables us to apply Fagin's threshold algorithm [11] to efficiently determine the k datasets having the highest g_1 score for the query W.

Location-aware score components The location – title (T), row or column headers (HR or HC), or comments (C) – where a keyword occurs in a dataset can also be used to assess the dataset relevance. For example, a keyword appearing in the title often signals a dataset more relevant for the search than if the keyword appears in a comment. We exploit this observation to *pre-filter* the datasets for which we compute exact scores, as follows.

We run the TA algorithm using the score function g_1 to select $r \times k$ datasets, where r is an integer larger than 1. For each dataset thus retrieved, we compute a second, *refined* score function g_2 (see below), which also accounts for the locations in which keywords appear in the datasets; the answer to the query will consists of the top-k datasets according to g_2.

The second score function $g_2(D,W)$ is computed as follows. Let w' be a word appearing at a location $loc \in \{\mathsf{T}, \mathsf{HR}, \mathsf{HC}, \mathsf{C}\}$ such that $w' \prec W$. We denote by $w'_{loc,D}$ (or just w'_{loc} when D is clear from the context) the *existence of one or several located occurrence of* w' *in* D *in* loc. Thus, for instance, if "youth" appears twice in the title of D and once in a row header, this leads to two located occurrences, namely $\text{youth}_{\mathsf{T},D}$ and $\text{youth}_{\mathsf{HR},D}$.

Then, for $loc \in \{\mathsf{T}, \mathsf{HR}, \mathsf{HC}, \mathsf{C}\}$ we introduce a coefficient α_{loc} allowing us to calibrate the weight (importance) of keyword occurrences in location loc. To quantify D's relevance for W due to its loc occurrences, we define a *location score component* $f_{loc}(D,W)$. In particular, we have experimented with two f_{loc} functions:

- $f^{sum}_{loc}(D,W) = \alpha_{loc}^{\sum_{w \prec W} score(w_{loc,D})}$
- $f^{count}_{loc}(D,W) = \alpha_{loc}^{count\{w \prec W\}}$

where for $score(w_{loc,D})$ we use the value $score(w)$, the score of D with respect to w (Section 4.4). Thus, each f_{loc} "boosts" the relevance scores of all loc occurrences by a certain exponential formula, whose relative importance is controlled by α_{loc}.

Further, the relevance of a dataset increases if different query keywords appear *in different header locations*, that is, some in HR (header rows) and some in HC (header columns). In such cases, the data cells at the intersection of the respective rows and columns may provide very precise answers to the query, as illustrated in Figure 3: here, "France" is present in HC while "youth" and "17" appear in HR. To reflect this, we introduce another function $f_{\mathrm{H}}(D,W)$ computed on the scores of all unique located occurrences from row or column headers; we also experimented with the two variants, f^{sum}_{H} and f^{count}_{H} introduced above.

Putting it all together, we compute the content- and location-aware relevance score of a dataset for W as:

$$g_2(D,W) = g_1(D,W) + \Sigma_{loc \in \{\mathsf{T},\mathsf{HR},\mathsf{HC},\mathsf{C}\}} f_{loc}(D,W) + f_{\mathsf{H}}(D,W)$$

Finally, we also experimented with another function $g^*(D,W)$ defined as:

$$g^*_2(D,W) = \begin{cases} g_2(D,W), \text{if } f_{\mathsf{T}}(D,W) > 0 \\ 0, \text{otherwise} \end{cases}$$

g^*_2 discards datasets having no relevant keyword in the title. This is due to the observation that statistic dataset titles are typically carefully chosen to describe the dataset; if no query keyword can be connected to it, the dataset is very likely not relevant.

4.6 Data cell search

We now consider the problem of identifying the data cell(s) (or the data rows/columns) that can give the most precise answer to the user query.

Such an answer may consist of exactly one data *cell*. For example, for the query "unemployment rate in Paris", a very good answer would be a data cell $D_{r,c}$ whose closest row header cell contains "unemployment rate" and whose closest column header cell contains "Paris". Alternatively, query keywords may occur not in the closest column header cell of $D_{r,c}$ but in another header cell that is its ancestor in D. For instance, in Figure 3, let $D_{r,c}$ be the data cell at the intersection of the Aug-17 column with the France row: the word "youth" occurs in an ancestor of the Aug-17 header cell, and "youth" clearly characterizes $D_{r,c}$'s content. We say the closest (row and column) header cells of $D_{r,c}$ and all their ancestor header cells *characterize* $D_{r,c}$.

Another valid answer to the "unemployment rate in Paris" query would be a whole data *row* (or a whole *column*) whose header contains "unemployment" and "Paris". We consider this to be less relevant than a single data cell answer, as it is less precise.

We term *data cell answer* an answer consisting of either a data cell, or a row or column; below, we describe how we identify such answers.

We identify data cells answers from a given dataset D as follows. Recall that all located occurrences in D, and in particular those of the form w_{HR} and w_{HC} for $w \prec W$, have been pre-computed; each such occurrence corresponds either to a header row r or to a header column c. For each data cell $D_{r,c}$, we define $\#(r,c)$ as the number of unique words $w \prec W$ occurring in the header cells characterizing $D_{r,c}$. Data cells in D may be characterized by:

(1) Some header cells containing HR occurrences (for some $w \prec W$), and some others containing HC occurrences;
(2) Only header cells with HR occurrences (or only header cells with HC ones).

Observe that if D holds both cell answers (case 1) and row- or column answers (case 2), by definition, they occur in different rows and columns. Our returned data cell answers from D are:

- If there are cells in case 1, then each of them is a data cell answer from D, and we return cell(s) with highest $\#(r,c)$ values.
- Only if there are no such cells but there are some relevant rows or columns (case 2), we return the one(s) with highest $\#(r,c)$ values. This is motivated by the intuition that if D has a specific, one-cell answer, it is unlikely that D also holds a less specific, yet relevant one.

Concretely, we compute the $\#(r,c)$ values based on the (word, score, location, dataset) tuples we extract (Section 4.4). We rely on SPARQL 1.1 [30] queries on the RDF representation of our datasets (Section 4.1) to identify the cell or row/column answer(s) from each D. SPARQL 1.1 features property paths, akin to regular expressions; we use them to identify all the header cells characterizing a given $D_{r,c}$.

Note that this method yields only one element (cell, row or column) from each dataset D, or several elements if they have the exact same score. An alternative would have been to allow returning several elements from each dataset; then, one needs to decide how to collate (inter-rank) scores of different elements identified in different datasets. We consider that this alternative would increase the complexity of our approach, for unclear benefits: the user experience is often better when results from the same dataset are aggregated together, rather than considered as independent. Suggesting several data cells per dataset is then more a question of result visualization than one pertaining to the search method.

5 EXPERIMENTAL EVALUATION

We describe here experiments evaluating the quality and the speed of our algorithms. Section 5.1 considers RDF data extraction from spreadsheet data, while Section 5.2 studies query answering.

5.1 Extraction

From 20,743 Excel files collected, we selected at random 100 unseen files to evaluate the reliability of the extraction process; these files contained a total of 2432 tables. To avoid very similar files in our evaluation, e.g., tables showing the same metric for distinct years, we required the first three letters in the name of each newly selected file, to be different from the first three letters in the names of all the files previously selected to be included in the test batch.

For these 100 files, we visually identified the header cells, data cells and header hierarchy, which we compared with those obtained from our extractor. We consider a table is "correctly extracted" when all these are pairwise equal; otherwise, the table is "incorrectly extracted". *We recorded* 91% *(or* 2214*) tables extracted correctly, and* 9% *(or* 218*) incorrectly extracted.* Most of the incorrect extractions were due to sheets with several tables (not just one). We plan to extend our system to also handle them correctly.

Overall, the extraction produced $2.68 \cdot 10^6$ row header cells, $122 \cdot 10^6$ column header cells, and $2.24 \cdot 10^9$ data cells; the extraction algorithm ran for 5 hours.

5.2 Search algorithm

In this section, we evaluate the quality and speed of our search algorithm. Section 5.2.1 describes the dataset and query workload we used, which was split into a development set (on which we tuned our score functions) and a test set (on which we measured the performance of our algorithms). It also specifies how we built a "gold-standard" set of answers against which our algorithms were evaluated. Section 5.2.2 details the choice of parameters for the score functions.

5.2.1 Datasets and Queries. We have developed our system in Python (61 classes and 4071 lines). Crawling the INSEE Web site, we extracted information out of **19,984** HTML tables and **91,161** spreadsheets, out of which we built a Linked Open Data corpus of **945 millions** RDF triples.

We collected all the articles published online by the fact-checking team "Les Décodeurs",[8] a fact-checking and data journalism team of Le Monde, France's leading national newspaper, between March 10th and August 2nd, 2014; there were 3,041 articles. From these, we selected 75 articles whose fact-checks were backed by INSEE data; these all contain links to https://www.insee.fr. By reading these articles and visiting their referenced INSEE dataset, we identified a set of 55 natural language queries which the journalists could have asked a system like ours.[9]

We experimented with a total of 288 variants of the g_2 function:

- g_1 was either g_{1a}, g_{1b} or g_{1c};
- g_2 relied either on f_{loc}^{sum} or on f_{loc}^{count}; for each of these, we tried different value combinations for the coefficients α_{T}, α_{HC}, α_{HR} and α_{C};
- we used either the g_2 formula, or its g_2^* variant.

We built a gold-standard reference to this query set as follows. We ran each query q through our dataset search algorithm for each of the 288 g_2 functions, asking for $k = 20$ most relevant datasets. We

[8] http://www.lemonde.fr/les-decodeurs/

[9] This was not actually the case; our system was developed after these articles were written.

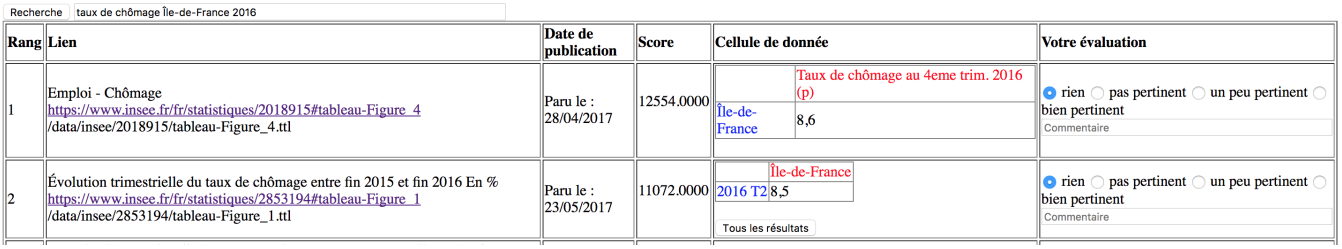

Figure 4: Screen shot of our search tool. In this example, the second result is a full column; clicking on "*Tous les résultats*" (all results) renders all the cells from that column.

built the union of all the answers thus obtained for q and assessed the relevance of each dataset as either 0 (not relevant), 1 ("partially relevant" which means user could find some related information to answer their query) or 2 ("highly relevant" which means user could find the exact answer for their query); a Web front-end was built to facilitate this process.

5.2.2 Search experiments. We specify our evaluation metric, then describe how we tuned the parameters of our score function, and the results of our experiments focused on the quality of the returned results. Last but not least, we put them into the perspective of a comparison with the baselines which existed prior to our work: INSEE's own search system, and plain Google search.

Evaluation Metric

We evaluated the quality of the answers of our runs and of the baseline systems by their mean average precision (MAP[10]), widely used for evaluating ranked lists of results.

MAP is traditionally defined based on a binary relevance judgment (relevant or irrelevant in our case). We experimented with the two possibilities:

- MAP_h is the mean average precision where only highly relevant datasets are considered as relevant
- MAP_p is the mean average precision where both partially and highly relevant datasets are considered relevant.

Parameter Estimation and Results We experimented with the following flavors of the g_1 function :

- $g_{1b}(D, W) = 10^{\sum_{w<W} score(w)}$
- $g_{1d}(D, W) = 10^{count\{w<W\}}$
- $g_{1f}(D, W) = \sum_{w<W} 10^{score(w)}$

We also experimented with some modified variants that take into account the sum of matching keywords:

- $g_{1c}(D, W) = \sum_{w<W} score(w) + g_{1b}(D, W)$
- $g_{1e}(D, W) = \sum_{w<W} score(w) + g_{1d}(D, W)$
- $g_{1g}(D, W) = \sum_{w<W} score(w) + g_{1f}(D, W)$

A randomly selected development set of 29 queries has been used to select the best values for the 7 parameters of our system : α_T, α_C, α_{HR}, α_{HC}, α_H, as well as the different versions of g_1 and g_2. For this purpose, we ran a grid search with different values of these parameters, selected among {3,5,7,8,10}, on the development query set, and applied the combination obtaining the best MAP results on the test set (composed of the remaining 26 queries).

[10]https://en.wikipedia.org/wiki/Information_retrieval#Mean_average_precision

	Dev. set 29 queries	Dev. set 17 queries
MAP_p	0.82	0.83
MAP_h	0.78	0.80

Table 1: Results on the first and second development set.

We found that a same score function has lead to the best MAP_h *and* the best MAP_p on the development query set. In terms of the notations introduced in Section 4.5, this best-performing score function is obtained by:

- Using $g_{1,c}$;
- Using f^{sum} and the coefficient values $\alpha_T = 10$, $\alpha_C = 3$, $\alpha_{HR} = 5$, $\alpha_{HC} = 5$ and $\alpha_H = 7$;
- Using the g_2^* variant, which discards datasets lacking matches in the title.

On the test set, this function has lead to $\boxed{MAP_p = 0.76}$ and $\boxed{MAP_h = 0.70}$.

Given that our test query set was relatively small, we performed two more experiments aiming at testing the robustness of the parameter selection on the development set:

- We used a randomly selected subset of 17 queries among the 29 development queries, and used it as a new development set. The best score function for this new development set was the same; further, the MAP results on the two development sets are very similar (see Table 1).
- We computed the MAP scores obtained on the full development set for all 288 combinations of parameters, and plotted them from the best to the worst (Figure 5; due to the way we plot the data, two MAP_h and MAP_p values shown on the same vertical line may not correspond to the same score function). The figure shows that the best-performing 15 combinations leads to scores higher than 0.80, indicating that any of these could be used with pretty good results.

These two results tend to show that we can consider our results as stable, despite the relatively small size of the query set.

Running time Processing and indexing the words (close to those) appearing in the datasets took approximately three hours. We ran our experiments on a machine with 126GB RAM and 40 CPUs Intel(R) Xeon(R) E5-2640 v4 @ 2.40GHz. The average query evaluation time over the 55 queries we identified is 0.218 seconds.

Comparison against Baselines To put our results into perspective, we also computed the MAP scores of our test query set on the

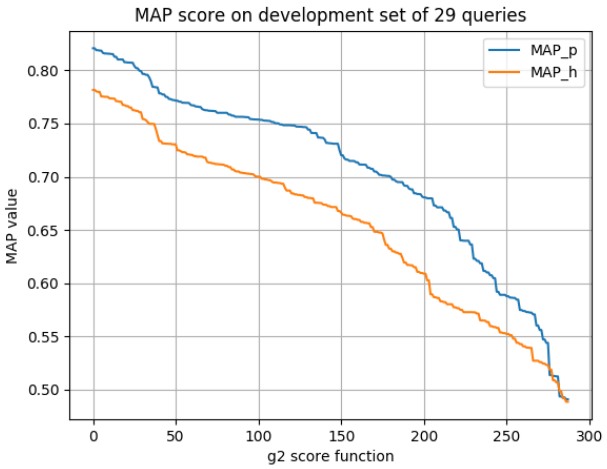

Figure 5: *MAP* results on the development set for 288 variants of the score function.

	Our system	INSEE search	Google search
MAP_p	0.76	0.57	0.76
MAP_h	0.70	0.46	0.69

Table 2: Comparing our system against baselines.

two baselines available prior to our work: INSEE's own dataset search system[11], and Google search instructed to look only within the INSEE web site. Similarly to the evaluation process of our system, for each query we selected the first 20 elements returned by these systems and manually evaluated each dataset's relevance to the given query. Table 2 depicts the MAP results thus obtained, compared against those of our system. Google's MAP is very close to ours; while our work is obviously not placed as a rival of Google in general, we view this as validating the quality of our results with (much!) smaller computational efforts. Further, our work follows a white-box approach, whereas it is well known that the top results returned by Google are increasingly due to other factors beyond the original PageRank [3] information, and may vary in time and/or with result personalization, Google's own A/B testing etc.

We end this comparison with two remarks. (*i*) Our evaluation was made on INSEE data alone due to the institute's extensive database on which fact-checking articles were written, from which we derived our test queries. However, as stated before, our approach could be easily adapted to other statistic Web sites, as we only need the ability to crawl the tables from the Web site. As is the case for INSEE, this method may be more robust than using the category-driven navigation or the search feature built in the Web site publishing the statistic information. (*ii*) Our system, based on a fine-granularity modeling of the data from statistic tables, is the only one capable of returning *cell-level answers* (Section 4.6). We show such answers to the users together with the header cells characterizing them, so that users can immediately appreciate their accuracy (as in Figure 4).

[11] Available at https://insee.fr

6 RELATED WORK

Section 6.1 puts our data extraction work into the perspective of comparable methods. Then, Section 6.2 focuses on answering keyword queries.

6.1 Extraction

Closest to our work, [7] automatically extracts relational data from spreadsheets. Their system used conditional random fields (CRF) [18] to identify *data frames*: such a frame is a *value region* which contains numeric cells, *top attributes* and *left attributes* corresponding to header cells and their hierarchies. From a collection of 410,554 Excel files collected from the Web, they selected randomly 100 files, labeled 27,531 non-empty rows (as title, header, data or footnote) manually by human experts, then used this dataset for training and evaluation. They obtained significant better precision and recall metrics when taking into account both textual and layout features for CRF. SVM was applied on header cells to learn about their hierachies. The model achieved a precision of 0.921 for top headers, and 0.852 for left headers. However, these numbers should not be compared with our results, because the datasets are different (tables encountered on the Web *vs.* government statistics), as well as the evaluation metrics (cell-by-cell assessment in [7] *vs.* binary assessment over the entire sheet extraction in our case). Overall, their method is more involved, whereas our method has the advantage of not requiring manual labeling. We may also experiment with learning-based approaches in the future. [1] describes an extractor to automatically identify in spreadsheets entities like location and time.

Data extraction from tables encoded as HTML lists has been addressed e.g., in [10]. Structured fact extraction from text Web pages has lead to the construction of knowledge bases such as Yago [22] and Google's Knowledge Graph [8]. Our work has a related but different focus as we focus on high-value, high-confidence, fine-granularity, somehow-structured data such as government-issue statistics; this calls for different technical methods.

In prior work [12], we devised a tool for exploiting background information (in particular, we used DBPedia RDF data) to add context to a claim to be fact-checked; in this work, we tackle the complementary problem of producing high-value, high-confidence Linked Open Data out of databases of statistics.

While in this work, we focused on improving the usability of statistic tables (HTML tables or spreadsheets) as reference data sources against which claims can be fact-checked, other works focused on building textual reference data source from general claims [2, 21], congressional debates [28] or tweets [25].

Some works focused on exploiting the data in HTML and spreadsheet tables found on the Web. Tschirschnitz et al. [29] focused on detecting the semantic relations that hold between millions of Web tables, for instance detecting so-called *inclusion dependencies* (when the values of a column in one table are included in the values of a column in another table).

6.2 Search problem

Also seeking to facilitate access to spreadsheet data, M. Kohlhase et al. [17] built a search engine for finding and accessing spreadsheets by their formulae. This is less of an issue for the tables we focus on, as they contain plain numbers and not formulas.

We are not aware of a previous work addressing the same question as ours, namely: finding the tabular dataset, and the data cell answers, most relevant to a given keyword query. Keyword search has been addressed by a vast literature in text and Web documents, as well as for XML documents, e.g. [16], relational databases, e.g. [26], and RDF graphs, e.g. [9, 19]. Thus, one could apply such an RDF keyword search algorithm to our search problem. However, there are differences: (*i*) the abovementioned RDF keyword search work do not return datasets, but answer trees, that is: trees that are subgraphs of the RDF graph, such that each tree has one node containing each query keyword. From this perspective, an RDF keyword search algorithm competes with our cell search but not with our dataset search; (*ii*) more importantly, answer trees to RDF keyword search are ranked according to metrics such as their size (number of edges), and such metrics are agnostic of the particular type and role of each RDF node. In the RDF datasets we work with, data and header cells play very specific roles, which are not taken into account by a generic RDF keyword search algorithm would not. In contrast, our approach ranks datasets, and then data cell answers, by carefully considering all their content (in title, headers, comments) and their spatial placement of such content in the original table, as discussed in Section 4.5 and Section 4.6.

Google's Fusion Tables work [13] focuses on storing, querying and visualizing tabular data, however, it does not tackle keyword search with a tabular semantics as we do. Google has also issued Google Tables as a working product[12]. In March 2018, we tried to use it for some sample queries we addressed in this paper, but the results we obtained were of lower quality (some were irrelevant). We believe this may be due to Google's focus on data available on the Web, whereas we focus on very high-quality data curated by INSEE experts, but which needed our work to be easily searchable.

Currently, our software is not capable of aggregating information, e.g., if one asks for unemployed people from all departements within a region, we are not capable of summing these numbers up into the number corresponding to the region. This could be addressed in future work which could focus on applying OLAP-style operations of drill-down or roll-up to further process the information we extract from the INSEE datasets.

Our extraction method was published in [4], while the query algorithm will be soon presented in [5].

7 CONCLUSION

We have described an ongoing effort to extract Linked Open Data from data tables produced by the INSEE statistics institute, and shared by them under the form of Excel tables. We do this as part of our work in the ContentCheck R&D project[13], which aims at investigating content management techniques (from databases, knowledge management, natural language processing, information extraction and data mining) to provide tools toward automating and supporting fact-checker journalists' efforts. Computational fact-checking is a very active areas, approached from many perspectives [6], [20]. In the effort described here, we worked toward enabling trusted statistic data sources to be used as background information, when trying to determine the degree of thruthfulness of a claim.

[12]https://research.google.com/tables

[13]http://team.inria.fr/cedar/contentcheck

Our current work aims at automatically identifying mentions of statistics within texts, for instance, in a phrase of the form "Despite many successive public policies, youth unemployment in France is still above 10%", our goal is to automatically identify "youth unemployment above 10%". Combined with the search functionality described in this work, such automatic identification would allow moving one step forward toward automating fact-checking.

Acknowlegements This work was supported by the Agence Nationale pour la Recherche (French National Research Agency) under grant number ANR-15-CE23-0025-01.

REFERENCES

[1] Ramoza Ahsan, Rodica Neamtu, and Elke Rundensteiner. 2016. Towards Spreadsheet Integration Using Entity Identification Driven by a Spatial-temporal Model. In *ACM SAC*. ACM, New York, NY, USA, 1083–1085.

[2] R. Bar-Haim, I. Bhattacharya, F. Dinuzzo, A. Saha, and N. Slonim. 2017. Stance Classification of Context-Dependent Claims. In *EACL*. pages 251–261.

[3] S. Brin and L. Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30, 1-7 (1998), 107–117.

[4] T.D. Cao, I. Manolescu, and X. Tannier. 2017. Extracting Linked Data from Statistic Spreadsheets. In *Proceedings of The International Workshop on Semantic Big Data (SBD), in conjunction with SIGMOD*.

[5] T.D. Cao, I. Manolescu, and X. Tannier. 2018. Searching for Truth in a Database of Statistics. In *Proceedings of The International Workshop on Web and Databases (WebDB), in conjunction with SIGMOD*.

[6] Sylvie Cazalens, Philippe Lamarre, Julien Leblay, Ioana Manolescu, and Xavier Tannier. 2018. A Content Management Perspective on Fact-Checking. In *The Web Conference 2018 - alternate paper tracks "Journalism, Misinformation and Fact Checking"*. Lyon, France, 1–10. https://hal.archives-ouvertes.fr/hal-01722666

[7] Zhe Chen and Michael Cafarella. 2013. Automatic Web Spreadsheet Data Extraction. In *Proceedings of the 3rd International Workshop on Semantic Search Over the Web (Semantic Search)*. ACM, New York, NY, USA, Article 1, 8 pages.

[8] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. 2014. From Data Fusion to Knowledge Fusion. *PVLDB* 7 (2014).

[9] Shady Elbassuoni and Roi Blanco. 2011. Keyword Search over RDF Graphs. In *ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, New York, NY, USA, 237–242. http://doi.acm.org/10.1145/2063576.2063615

[10] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. 2011. Harvesting Relational Tables from Lists on the Web. *The VLDB Journal* 20, 2 (April 2011), 209–226.

[11] R. Fagin, A. Lotem, and M. Naor. 2003. Optimal Aggregation Algorithms for Middleware. *J. Comput. Syst. Sci.* 66, 4 (June 2003), pages 614–656.

[12] François Goasdoué, Konstantinos Karanasos, Yannis Katsis, Julien Leblay, Ioana Manolescu, and Stamatis Zampetakis. 2013. Fact Checking and Analyzing the Web (demonstration). In *ACM SIGMOD*.

[13] H. Gonzalez, A. Halevy, C. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen. 2010. Google Fusion Tables: Data Management, Integration, and Collaboration in the Cloud. In *SOCC*.

[14] Jonathan Gray, Lucy Chambers, and Liliana Bounegru. 2012. *The Data Journalism Handbook: How Journalists can Use Data to Improve the News*. O'Reilly.

[15] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 2007. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *CoRR* abs/cs/0701155 (2007).

[16] V. Hristidis and Y. Papakonstantinou. 2002. DISCOVER: Keyword Search in Relational Databases. In *Very Large Databases Conference (VLDB)*. 670–681.

[17] M. Kohlhase, C. Prodescu, and C. Liguda. 2013. XLSearch: A Search Engine for Spreadsheets. *EuSpRIG* (2013).

[18] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 282–289.

[19] Wangchao Le, Feifei Li, Anastasios Kementsietsidis, and Songyun Duan. 2014. Scalable Keyword Search on Large RDF Data. *IEEE TKDE* 26, 11 (2014).

[20] Julien Leblay, Ioana Manolescu, and Xavier Tannier. 2018. Computational fact-checking: problems, state of the art, and perspectives. In *The Web Conference (tutorial)*.

[21] R. Levy, Y. Bilu, D. Hershcovich, E. Aharoni, and N. Slonim. 2014. Context Dependent Claim Detection. *COLING* (2014), pages 1489–1500.

[22] Farzaneh Mahdisoltani, Joanna Biega, and Fabian M. Suchanek. 2015. YAGO3: A Knowledge Base from Multilingual Wikipedias. In *CIDR*.

[23] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, and J. Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*. pages 3111–3119.

[24] Ř. Radim and S. Petr. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. pages 45–50.

[25] A. Rajadesingan and H. Liu. 2014. Identifying Users with Opposing Opinions in Twitter Debates. In *International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction*. pages 153–160.

[26] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan, and Luis Gravano. 2007. Efficient Keyword Search Across Heterogeneous Relational Databases. In *IEEE International Conference on Data Engineering (ICDE)*. 346–355. DOI: http://dx.doi.org/10.1109/ICDE.2007.367880

[27] The World Wide Web Consortium (W3C). 2014. Best Practices for Publishing Linked Data. Available at: https://www.w3.org/TR/ld-bp/. (2014).

[28] M. Thomas, B. Pang, and L. Lee. 2006. Get out the vote: Determining support or opposition from Congressional floor-debate transcripts. In *EMNLP*. 327–335.

[29] F. Tschirschnitz, T. Papenbrock, and F. Naumann. 2017. Detecting Inclusion Dependencies on Very Many Tables. *ACM Trans. Database Syst.* (2017), pages 18:1–18:29.

[30] W3C. 2008. SPARQL Protocol and RDF Query Language. http://www.w3.org/TR/rdf-sparql-query. (2008).

[31] You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2014. Toward Computational Fact-Checking. *PVLDB* 7, 7 (2014), 589–600.