The final publication is available at ACM via http://dx.doi.org/10.1145/3342558.3345409

XLIndy: Interactive Recognition and Information Extraction in Spreadsheets

Elvis Koci
elvis.koci@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Maik Thiele
maik.thiele@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Julius Gonsior
julius.gonsior@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

Oscar Romero
oromero@essi.upc.edu
Universitat Politècnica de Catalunya-BarcelonaTech
Barcelona, Spain

Wolfgang Lehner
wolfgang.lehner@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

ABSTRACT

Over the years, spreadsheets have established their presence in many domains, including business, government, and science. However, challenges arise due to spreadsheets being partially-structured and carrying implicit (visual and textual) information. This translates into a bottleneck, when it comes to automatic analysis and extraction of information. Therefore, we present XLIndy, a Microsoft Excel add-in with a machine learning back-end, written in Python. It showcases our novel methods for layout inference and table recognition in spreadsheets. For a selected task and method, users can visually inspect the results, change configurations, and compare different runs. This enables iterative fine-tuning. Additionally, users can manually revise the predicted layout and tables, and subsequently save them as annotations. The latter is used to measure performance and (re-)train classifiers. Finally, data in the recognized tables can be extracted for further processing. XLIndy supports several standard formats, such as CSV and JSON.

CCS CONCEPTS

• **Information systems** → *Enterprise information systems*; *Document structure*; *Information extraction*; Mediators and data integration.

KEYWORDS

spreadsheets, table recognition, layout inference, information extraction, visualization, annotation, interactive, excel, add-in

ACM Reference Format:
Elvis Koci, Maik Thiele, Julius Gonsior, Oscar Romero, and Wolfgang Lehner. 2019. XLIndy: Interactive Recognition and Information Extraction in Spreadsheets. In *Proceedings of The 19th ACM Symposium on Document Engineering (DocEng 2019).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

DocEng 2019, September 23–26, Berlin, Germany

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Spreadsheets are intuitive to use and highly effective data processing tools. As such, they have been adopted for various tasks in many different settings. In business, spreadsheets are frequently used for financial analysis and reporting [7]. Governmental institutions use them to collect and make data available (e.g., in open data platforms [3]). In science, spreadsheets act as lab books, or even as sophisticated calculators and simulators [16].

Seeing this wide usage and the concentration of valuable data in spreadsheets, industry and research have recognized the need for automatic processing. Recent attempts, such as *Ideas* in Excel and *Explore* in Google Sheets, aim at providing insights and recommendations to users (e.g., summary statistics and charts), based on background analysis of tabular data in the sheet. Other works [1, 3, 5, 17], including ours [10–15], focus on integrating and extracting data from spreadsheets. One of the main concerns comes with data and knowledge being scattered in multiple spreadsheet files. This can lead to information silos, unless appropriate methods are put in place to process these data automatically. Such methods can enhance interoperability, encourage better use, and enable more control over spreadsheet data.

However, spreadsheets are designed primarily for human consumption, and as such they favor customization and visual comprehension. Data are often intermingled with formatting artifacts and textual metadata, which carry domain-specific or even user-specific information (i.e., personal preferences). Multiple tables, with different layout and structure, can be found in the same sheet. Moreover, these tables do not adhere to a predefined formal data model. Altogether, spreadsheets are better described as partially-structured documents, with a significant degree of implicit information.

In this paper, we present XLIndy, which showcases our innovative approaches for tackling some of the aforementioned challenges. The front-end of XLIndy is developed as a Microsoft Excel add-in. We re-utilize the powerful native user interface, while introducing custom panels and menus when needed. The results from the implemented approaches are visually displayed to the user, in order to facilitate fast inspection, comparison, and fine-tuning. Furthermore, the add-in plays the role of a mediator between the native

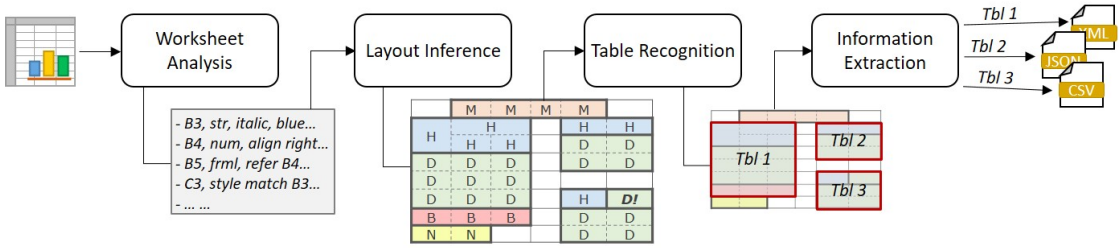

Figure 1: Processing pipeline

Excel application and the machine learning back-end. The latter holds utilities and the proposed approaches, which were entirely developed in Python. In this way, we take advantage of the many available and highly efficient Python libraries.

Contributions. In literature we find existing systems attempting information extraction in spreadsheets [4, 6]. Their main objective is to extract and load normalized spreadsheet data into a relational database. Clearly, this enables many possible uses for spreadsheet data. However, these works cover only a small subset of scenarios that might occur in spreadsheets. Instead, we assume any number of tables, with arbitrary arrangement, exhibiting high variability in contents, formatting, and layout. XLIndy is not an end-user product, but rather a framework for testing and interacting with our innovative approaches. Additionally, users can give their feedback (e.g., repairing results), which is used for further improvement. Ultimately, XLIndy provides some support for information extraction, in formats such as CSV and JSON. This already facilitates the re-use of spreadsheet data for the majority of information systems.

Paper Outline. In the following section, we discuss the proposed processing pipeline, and its individual steps. We provide a high-level overview of the XLIndy system, in Section 3. Finally, in Section 4, we outline the expected demonstration of our tool.

2 PROCESSING PIPELINE

We see recognition and information extraction in spreadsheets as a series of steps, which collectively form our processing pipeline, illustrated in Figure 1. Although we cover various aspects of automatic spreadsheet processing, our research focuses mainly on two crucial tasks: layout inference [13, 15] and table identification [10–12, 14]. Subsequently, we adapt approaches from related work, to extract the information from the detected tables.

2.1 Worksheet Analysis

For each *non-empty*[1] cell in the sheet, we collect a variety of features, detailed at [15]. We consider style features, like the font size, alignment, and background color. As well as, we study cell contents, addressing aspects such as value type, length, and special characters/words. Additionally, we pay close attention to formulas, a key spreadsheet functionality, but to some extent overlooked by related work. Here, we capture references (i.e., dependencies between cells), and determine the formula type (e.g., numeric aggregations). Moreover, we investigate the surroundings of a cell, measuring the similarity with its neighbors, in terms of style and content.

[1]It contains a value, which is not entirely made of white space characters.

2.2 Layout Inference

In simple terms, layout inference is the task of segmenting the contents of a document into regions of interest. Typically, the segmentation is based on both geometrical and logical analysis. In this work, we start by inferring the logical layout role of each *non-empty* cell, in a sheet. Subsequently, we group adjacent cells of the same role to form strictly rectangular layout regions (LRs). This second step largely depends on the geometry and the spatial arrangement of non-empty cells. Moreover, note that overall approach can describe arbitrary layouts. It foresees that vertical (tall) and horizontal (wide) LRs can coexist in the same sheet.

Figure 1, illustrates the layout regions for an example sheet. Bold lines indicate borders of LRs, while dashed lines separate the enclosed cells, which share the same layout role. We define seven such roles[2], some shown in Figure 1. *Header* (H) and *Data* (D) are the main ingredients of tables. *Notes* (N) and *Titles* (M) provide meta-information about specific areas or the whole sheet. Cells marked as *Derived* (B) hold numeric aggregations and summary values. *Attributes* describe cells participating in a parent-child relationship (i.e., hierarchy). While, *Other* is a placeholder for everything else.

To infer these layout roles, we use classifiers, which were trained in a supervised fashion. As outlined in [15], we tested five different machine learning algorithms. Based on our evaluation, Random Forest achieves the highest overall accuracy. Particularly, for *Data* and *Header*, the accuracy is consistently above 90%.

Nevertheless, misclassifications (e.g., the cell marked as ***D!***) can lead to inconsistencies in the inferred layout. At [13] we propose an optional post-processing step, to tackle this challenge (not illustrated in Figure 1). Briefly, this step infers and repairs inconsistencies, taking into account the context from the surrounding layout regions. It considers their label, size, distance, and alignment with the currently in-focus LR.

2.3 Table Recognition

In this task, we attempt to identify areas of the sheet that hold tabular data. This demands an approach with high precision, since even the slightest mistakes, by a row or a column, could lead to false interpretations of the detected tables.

The inferred layout provides a good start for table recognition. However, we still need to tackle a series of challenges. Multiple tables and non-tables (e.g., lists, comments) can coexist in the same sheet. Also, they can be arranged in arbitrary ways, often using both directions: vertical (top-bottom) and horizontal (left-right).

[2]Previous publications use five roles. Here, we refine some and introduce new ones.

The tables themselves exhibit high diversity in terms of layout and structure. Additionally, empty cells, rows, and columns might be present inside the tables. Often, they indicate missing values or formatting artifacts (e.g., visual padding of contents). However, they can be easily misinterpreted, ultimately leading to information loss.

We have proposed several approaches for table recognition in spreadsheets [10, 12, 14]. Initially we employed heuristic- and rule-based methods. Most recently, we adopted genetic-based search and optimization techniques. The last two works, use a graph model to represent the overall layout of the sheet (i.e., the spatial arrangement of the inferred layout regions). Subsequently, the recognition task is formulated as a graph partitioning problem. We look for the optimum partitioning, such that the resulting subgraphs correspond precisely to true tables or non-tables of the sheet.

2.4 Information Extraction

XLIndy supports information extraction from recognized tables. We adapt approaches from related work [8, 17], to analyze tables and infer their schema. Initially, we check for hierarchies on the top rows and the left columns of the table. For instance, in Figure 3, the Header cells of the bottom table are stacked, spanning multiple rows. Clearly, the ones in the lower rows depend on those above. Furthermore, in the left column, months are grouped by quarter (i.e., Attributes). Having inferred hierarchies, we proceed to extract tuples. We traverse the top, and then move to the left hierarchies. An example tuple would be: *['Monitor', '1st Quarter', 'April', 421].*

3 SYSTEM OVERVIEW

XLIndy carries out the whole processing pipeline, described in the previous section. The users interact with the tool via a familiar interface, i.e., the Excel desktop application. On top of that, we deploy a custom add-in, which was developed in C#. This add-in triggers the execution of tasks from the pipeline, and subsequently handles the results. Nevertheless, the tasks themselves are performed by Python scripts, which stay at the back-end of the system. While, a local Python service (described in more details in the following sections) manages the communication between the front- and back-end.

Another aspect of the system is the physical layer, which holds spreadsheet, configuration, and temporary run-time files. Here, resides also the gold standard dataset [11], which is not a direct part of the system, but still tightly related to it. After all, on this dataset we train/test the proposed layout inference and table recognition methods. Furthemore, with the help of XLIndy, we can expand the gold standard with new annotated sheets, as outline in Section 3.4.

3.1 Back-end Processes

In an attempt to improve performance and cohesion in the system, we introduce a local Python service, which sits in between the front- and back-end components. One of its roles is to prefetch most of the required code, including custom modules, existing libraries, and serialized (layout inference) classifiers. This improves the execution of time of the developed methods, and gives a more interactive feeling to the user. Nevertheless, during run-time the operations are performed on-demand, triggered by the client add-in. The local service, which is running on the background, receives and then directs the call to the appropriate Python script. Subsequently, it returns the output of the script back to the add-in, which then displays the results to the user.

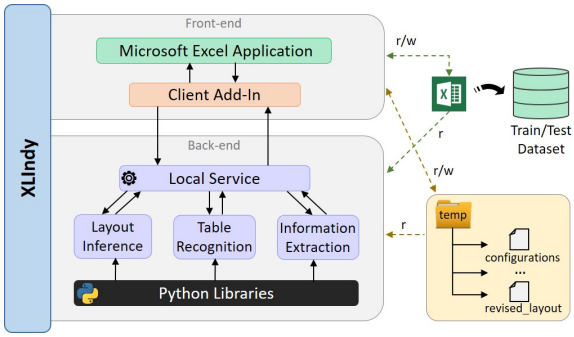

Figure 2: Architecture of XLIndy

3.2 Interoperability

At the moment, the .NET Framework has some support[3] for Python code. However, it lacks many features and libraries that come with other more popular Python implementations. Therefore, a workaround is to run Python scripts within processes initiated from C# code. Subsequently, the standard output of these processes is parsed, and then displayed to the user.

On the other side, the front-end sends data to the Python scripts, as well. However, operating systems usually impose limitations on the number and memory-size of the arguments passed to processes. To overcome these limitations, for some operations, we use temporary files to store data coming from the front-end. Then, via the local service we instruct the Python scripts to read these files.

3.3 Efficient Reading

The active Excel document is processed by both ends of the system. The front-end handles various operations which require read and/or write permissions. However, read permissions are sufficient for the back-end, since it only needs to collect features from the active sheet. To ensure consistency, if there are updates, the front-end will save the Excel file before making a call to the local service.

Occasionally, we get large documents. Therefore, the back-end employs an efficient reading mechanism. The .xlsx format is in reality a zipped directory[4] of XML files, which carry information regarding the cell values, formatting, references, and many other (detailed in the OOXML standard). Therefore, using the *openpyxl*[5] library, we read only XML files needed for the current task.

3.4 User Experience

Figure 3 provides a brief look at the user interface. It captures the state of the tool after the layout inference, and right before table recognition. Using build-in shape objects provided by Excel, XLIndy displays layout regions as colored coded rectangles, overlaying non-empty cells of the sheet. In the illustrated example, there is a misclassification, which the user repairs via the context menu.

[3] https://ironpython.net/
[4] http://officeopenxml.com/anatomyofOOXML-xlsx.php
[5] https://openpyxl.readthedocs.io/en/stable/

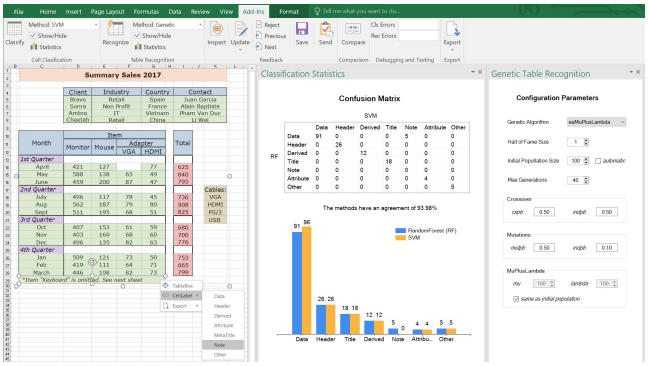

Figure 3: UI of the XLIndy tool

At the top Figure 3, we can see a custom ribbon, which acts as the primary menu for XLIndy. From here, the user carries the majority of the provided functionalities. Starting from the left, "Cell Classification" and "Table Recognition" sections allow users to initiate inference and recognition tasks, respectively. The user selects one of the several supported methods from the dropdown list, and depending on the desired task he/she clicks "Classify" or "Recognize". Once the results are displayed, the user can click the "Statistics" button, which opens a custom pane providing an overview of the performance (see Figure 3). The "Feedback" section of the ribbon helps the user navigate the results and revise them, if needed. After the user has reviewed the results, he/she can save them as annotations. These are stored in a special hidden sheet of the current document, allowing future reuse [11]. Moreover, from the ribbon, the user can compare different runs, via the "Compare" button. This will point to the differences, and emphasize the strengths and weaknesses of each run. Finally, the "Export" button dumps data from the selected table areas, in one of the supported formats.

Other panes allow the user to change configurations, before executing a task. In this way, users can fine-tune the selected approach, ultimately achieving the desired accuracy. Figure 3 displays the configuration pane for the genetic-based table recognition [14]. This pane opened automatically, when the user selected the "Genetic" method from the drop down list in the ribbon.

Furthermore, XLIndy supports a series of actions via a custom context menu. This makes certain operations straight forward. For example, in Figure 3 the user changes the role of a layout region from *Data* to *Derived*. Moreover, from this context menu, the user can manually create a tablebox (i.e., a rectangle shape with no fill), to indicate a tabular area. In this way, he/she can correct the table recognition results. Subsequently, the user can export data from tables, using the available options from this menu.

4 DEMONSTRATION WALKTHROUGH

Our presentation will introduce the functionality of XLIndy with a walkthrough example, as shown here[6]. The audience will load one of the available spreadsheets, and interact with the user interface, to complete the whole processing pipeline (detailed in Section 2).

[6]https://github.com/ddenron/xlindy_walkthrough

However, we will encourage the audience to experiment with different spreadsheets, from our gold standard dataset [11, 15]. We cover spreadsheets from companies [9], and others retrieved from the Web [2]. In this way, the audience can experience first hand the high variability in spreadsheet contents, and the challenges that come with it. Additionally, they can test and discover the pros and cons of the approaches that we propose. To this extend, they will use the available tools to analyze/compare results and change configurations (i.e., iterative fine-tuning).

Certainly, we give the audience the possibility to test their own Excel files, and ask in-depth questions about the architecture of the system and the implemented approaches.

REFERENCES

[1] Marco D Adelfio and Hanan Samet. 2013. Schema extraction for tabular data on the web. *Proceedings of the VLDB Endowment* 6, 6 (2013), 421–432.

[2] Titus Barik, Kevin Lubick, Justin Smith, John Slankas, and Emerson Murphy-Hill. 2015. Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets. In *the 12th Working Conference on Mining Software Repositories*. IEEE, 486–489.

[3] Zhe Chen and Michael Cafarella. 2013. Automatic web spreadsheet data extraction. In *International Workshop on Semantic Search over the Web*. ACM, 1.

[4] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. 2013. Senbazuru: A prototype spreadsheet database management system. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1202–1205.

[5] Zhe Chen, Sasha Dadiomov, Richard Wesley, Gang Xiao, Daniel Cory, Michael Cafarella, and Jock Mackinlay. 2017. Spreadsheet property detection with rule-assisted active learning. In *the International Conference on Information and Knowledge Management (CIKM)*. ACM, 999–1008.

[6] Julian Eberius, Christoper Werner, Maik Thiele, Katrin Braunschweig, Lars Dannecker, and Wolfgang Lehner. 2013. DeExcelerator: a framework for extracting relational data from partially structured documents. In *the International Conference on Information and Knowledge Management (CIKM)*. ACM, 2477–2480.

[7] Wayne W Eckerson and Richard P Sherman. 2008. Q&A: Strategies for Managing Spreadmarts. *Business Intelligence Journal* 13, 1 (2008), 23.

[8] David W Embley, Sharad Seth, and George Nagy. 2014. Transforming web tables to a relational database. In *the 22nd International Conference on Pattern Recognition*. IEEE, 2781–2786.

[9] Felienne Hermans and Emerson Murphy-Hill. 2015. Enron's spreadsheets and related emails: A dataset and analysis. In *the 37th IEEE/ACM International Conference on Software Engineering*, Vol. 2. IEEE, 7–16.

[10] Elvis Koci, Maik Thiele, Wolfgang Lehner, and Oscar Romero. 2018. Table recognition in spreadsheets via a graph representation. In *the 13th IAPR International Workshop on Document Analysis Systems (DAS)*. IEEE, 139–144.

[11] Elvis Koci, Maik Thiele, Josephine Rehak, Oscar Romero, and Wolfgang Lehner. 2019. DECO: A Dataset of Annotated Spreadsheets for Layout and Table Recognition. In *the 15th IAPR International Conference on Document Analysis and Recognition. (In Press)*.

[12] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. 2017. Table identification and reconstruction in spreadsheets. In *the International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 527–541.

[13] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. 2019. Cell Classification for Layout Recognition in Spreadsheets. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K '16: Revised Selected Papers)*, Ana Fred, Jan Dietz, David Aveiro, Kecheng Liu, Jorge Bernardino, and Joaquim Filipe (Eds.). Communications in Computer and Information Science, Vol. 914. Springer, Cham, 78–100.

[14] Elvis Koci, Maik Thiele, Oscar Romero, and Wolfgang Lehner. 2019. A Genetic-based Search for Adaptive Table Recognition in Spreadsheets. In *the 15th IAPR International Conference on Document Analysis and Recognition. (In Press)*.

[15] Elvis Koci, Maik Thiele, Óscar Romero Moral, and Wolfgang Lehner. 2016. A machine learning approach for layout inference in spreadsheets. In *IC3K 2016: The 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management: volume 1: KDIR*. SciTePress, 77–88.

[16] Cliff T Ragsdale. 2004. *Spreadsheet modeling and decision analysis*. Thomson south-western.

[17] Alexey O Shigarov and Andrey A Mikhailov. 2017. Rule-based spreadsheet data transformation from arbitrary to relational tables. *Information Systems* 71 (2017), 123–136.