Towards Automating Relational Data Wrangling

Gust Verbruggen and Luc De Raedt

Department of Computer Science, KU Leuven
gust.verbruggen@cs.kuleuven.be
luc.deraedt@cs.kuleuven.be

Abstract. It is well-known in data science that 80% of the work is devoted to preprocessing and only 20% to the actual machine learning or data mining step. This motivates us to explore different ways to (help) automate that preprocessing step. This note focusses on the question whether it is possible to (help) automate the data wrangling process for tabular data in data science.

1 Introduction

One long term goal of automatic machine learning and data science is to empower naive end-users to automatically analyse their data. Today we are far away from reaching that goal. In this note we explore how to help end-users with getting their data in the right format for analysis. As non-experts often gather their data in spreadsheets, we focus on the question whether it is possible to take such a spreadsheet and to automatically transform it into a format that can be used by standard machine learning software such as Weka [4]. Thus we want to help fully automating the data wrangling process [1].

Several approaches for data wrangling already exist. For example, the Wrangler [7] system provides an interactive interface for creating transformation programs without needing to write code. Instantiations of the FlashMeta [8] framework allow synthesising many data transformation programs by providing the system with input-output examples. A notable instantiation is FlashRelate [3] which allows for extracting relational data from spreadsheets. More recently, Foofah [6] aimed at a combination of these two: transforming a spreadsheet based on examples.

In this note we want to take the next step in these developments and explore whether these processes can be automated while focussing on data in tabular form. Relational data in tabular form has distinct properties that we exploit in order to mediate the need for examples describing the desired output. In contrast to previous approaches – where the subset of desired relational data is described by examples or intent – we aim to extract all relational data from the spreadsheet itself. We provide an initial approach to tackle this problem.

2 Problem

In this section we formulate the problem of wrangling relational data from spreadsheets that we focus on.

2.1 Spreadsheet Notation and Properties

We only consider spreadsheets that contain one table, but any rectangle finder can be used to extract multiple tables from spreadsheets. A spreadsheet with m columns and n rows is naturally represented by an $n \times m$ matrix in which each element is a cell.

Each cell has a known and given **type**. Because semantics of types aren't used, types are simply labels such as *integer* and *string* or just natural numbers. Empty cells get a distinct type $\varnothing$. The types are stored in a separate matrix.

2015	**OCT**	**NOV**	**DEC**
Hot			
Coffee	305	340	480
Tea	205	260	255
Hot Chocolate	301	364	470
Cold			
Fanta	103	164	101
Ice Tea	181	129	133
Coke	147	120	96
Coke Light	191	162	119
Orange Juice	102	168	103
Beer			
Stella Artois	601	573	951
Duvel	99	120	179

(a) Spreadsheet with beverage sales data for the fourth quarter of 2015. Similar spreadsheets exist for all quarters over different years.

year	**MTH**	**MTH**	**MTH**
type			
drink	AMT	AMT	AMT
drink	AMT	AMT	AMT
drink	AMT	AMT	AMT
type			
drink	AMT	AMT	AMT
drink	AMT	AMT	AMT
drink	AMT	AMT	AMT
drink	AMT	AMT	AMT
drink	AMT	AMT	AMT
type			
drink	AMT	AMT	AMT
drink	AMT	AMT	AMT

(b) Type spreadsheet for Table 1a.

Fig. 1

An m-ary relation $R \subseteq (A_1, \ldots, A_m)$ of n tuples can be easily embedded in a spreadsheet, represented by an $n \times m$ matrix. Each tuple simply becomes a row in the matrix and each attribute corresponds to a type.

Conversely, any spreadsheet S can be converted to a relation R_S by constructing tuples from the rows. Each column becomes an attribute of the relation. We can see that the resulting relation will only be meaningful if each column only consists of elements of the same type. Such a column is called **type-consistent** and a spreadsheet is type-consistent if all of its columns are.

2.2 Spreadsheet Transformation Programs

We limit ourselves to transformations that change the layout of the spreadsheet as in [5]. Such a transformation takes a matrix, optionally some arguments, and returns a new matrix which contains the same elements but repositioned, replicated or removed.

An example is the FOLD transformation, which takes a set of column indices as argument and folds them into one column by adding rows and using the column headers to indicate where it came from.

$$\textsc{fold}\left(\begin{bmatrix} & \mathbf{1} & \mathbf{2} \\ A & x & i_1 & j_1 \\ B & & i_2 & j_2 \\ C & & & j_3 \end{bmatrix}; \{2,3\}\right) = \begin{bmatrix} A & x & \mathbf{1} & i_1 \\ A & x & \mathbf{2} & j_1 \\ B & & \mathbf{1} & i_2 \\ B & & \mathbf{2} & j_2 \\ C & & \mathbf{1} & \\ C & & \mathbf{2} & j_3 \end{bmatrix}$$

The list of all such transformations that we consider to construct spreadsheet transformation programs is given in Table 1. It is heavily inspired from those used by existing approaches [7,6,9]. We can extend this list of transformations in order to support more complex spreadsheets.

Applying a transformation results in a **reconstruction error**. This is a measure of how well it can be inverted given its arguments; or how much information is lost when applying the transformation on an $n \times m$ spreadsheet. For example, the FOLD transformation has a reconstruction error of zero as it can be perfectly inverted, while DELETE has a reconstruction error that depends on of the number of non-empty cells it removes. The reconstruction errors for all operations are given in Table 1.

A spreadsheet transformation program $\mathcal{P}$ is then an ordered list of transformations $(t_1, \ldots, t_a)$ that are applied in order on a spreadsheet S such that $\mathcal{P}(S) = t_a(\ldots t_1(S)\ldots)$. The reconstruction error of a program is the sum of reconstruction errors of its transformations.

2.3 Problem

Given a spreadsheet S and a set of spreadsheet transformation operators, the task is to learn a program $\mathcal{P}$ over the operators such that $\mathcal{P}(S)$ is type-consistent while keeping the reconstruction error caused by $\mathcal{P}$ at 0.

We start from the assumption that a relation R was embedded in spreadsheet space and then restructured using a transformation program over an unknown set of transformations. The result is a spreadsheet S for which R_S is not equal to R, but for which we know that R can still be extracted by recovering the structure. A program that successfully transforms an arbitrary spreadsheet into a type-consistent one is believed to have uncovered this underlying structure. Keeping reconstruction error at 0 is necessary to discourage trivial results. For example, only keeping one arbitrary row from the spreadsheet will always result in type-consistency, but will probably not be a desired relation.

As we are only given an input example, this can be seen as a form of predictive program synthesis. The desired output is not given but predicted together with the program that produces it by using the type-consistency constraint, similar to using delimiter alignment in predicting text splitting tasks [10]. Strong enforcement of the type-consistency constraint results in rectangular output tables, another property of relational tables.

Simplifications In these first experiments some simplifications are taken into account. We assume that the types are known up to a granularity that distinguishes each attribute of the relation. For example, a binary relation having types *int* and *int* is not valid, but types *Age* and $Weight$ or 1 and 2 are. The input spreadsheet furthermore contains no noisy, nor missing values.

3 Predictive Program Synthesis

This section explains the naïve algorithm used to perform predictive synthesis. First, we introduce an heuristic that allows to perform a greedy search. Next, this algorithm is presented.

3.1 Heuristic

The type-consistency constraints alone is not useful when searching the program space. We introduce a measure of how well the constraint is satisfied that can serve as a heuristic. Three main properties are taken into account.

First, columns that contain different types need to be punished. Empty values are a wildcard because they are not necessarily wrongly typed. For example, type might still have to be made explicit by a forward fill. This leads to the type-consistency $\textsc{tc}_c$ of a column c to be defined as the joint proportion of the most occurring type and empty type over the total number of elements in the column.

Second, empty cells are addressed by rescaling $\textsc{tc}_c$ with the inverse proportion of missing values $\textsc{m}_c$. While these first two properties already result in fully type-consistent tables, they don't take the shape of the table into account. A table with two columns of the same type will get the maximal heuristic value, but it still violates the fact that each column has to contain a distinct attribute.

Finally, let the type of a column be the type that occurs most often in that column. We scale the heuristic a second time by the proportion of unique column values $\textsc{u}$. The heuristic value for a table T_S with m columns then becomes

$$H(T_S) = \left(\frac{1}{m} \sum_{c=1}^{m} \textsc{tc}_c (1 - \textsc{m}_c) \right) \textsc{u}. \tag{1}$$

We get $H(T_S) = 1$ for a table T_S that satisfies all constraints, providing a stopping criterium for the algorithm.

3.2 Naïve Algorithm for Predictive Synthesis

As we have a heuristic that we can optimise and a constraint to be satisfied, a naïve approach for predictive synthesis is a best-first search until the constraint is satisfied.

The parameter space for each transformation can be pruned based on the current table. Most transformations have a computable set of parameters that are *useful* to apply on a certain table. For example, if no empty values exist in a

column, forward filling that column has no effect. Similarly, when folding some columns together we can assume their headers to be of the same type as they will be grouped in one column. Table 1 lists the pruned list of possible arguments for each transformations.

The naïve synthesis algorithm for a table T_S is then a very simple greedy search. As long as no program $\mathcal{P}$ is found for which $H(\mathcal{P}(T_S)) = 1$, the best program so far is extended with all useful transformations.

4 Results

Assume a bar has gathered spreadsheets as in Table 1a over the course of a few years. They now want to get insights in their data such as correlations between the month and sales of hot beverages or a prediction of sales. These are easy tasks that can be performed using e.g. Weka – given that the data is in an appropriate format. Unfortunately, it will not be able to handle the raw data from Table 1a.

Running the naïve synthesis algorithm on this spreadsheet proceeds like the search tree in Figure 2. It can be seen that parameter pruning is highly effective at only trying parameter sets that *make sense*. The desired STP

Fold(1, 3)
Delete(0)
Split(0)
ForwardFill(1)
Delete(0)

is found without backtracking. Running this program on Table 1a results in a relation

`(year, type, drink, month, amount)`.

Different tables for all quarters can be easily joined. The results can be used by standard tools as the relation could be stored in a simple `.csv` file.

Other simple spreadsheets, based on real world spreadsheets such as those from the Fuse [2] corpus, are similarly wrangled correctly into a relational format. An additional use case is shown in Appendix 1.B.

5 Comparison to other approaches

In general, by trying to extract all relational information rather than a subset, we aim to wrangle relational data from spreadsheets without any user interaction. Previous approaches require some manual effort, but allow for a subset of the output space to be extracted. In the context of automated data analysis this tradeoff in flexibility is reasonable, if not beneficial, as features extraction and/or construction will still take place.

Wrangler enables the user to synthesise transformation programs by visually specifying regions of the spreadsheet that are to be fixed, for which an

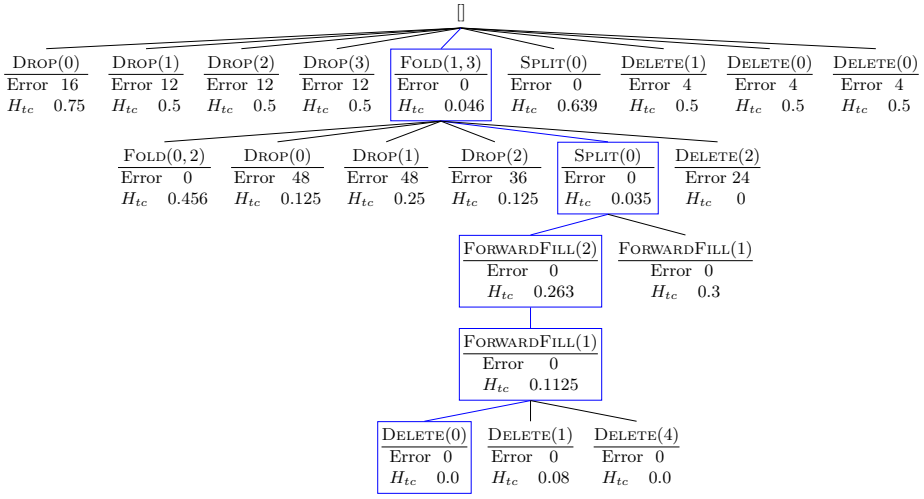

Fig. 2: Search tree for running the naïve synthesis algorithm on the spreadsheet in Table 1a. After the second level, only possible transformations with 0 reconstruction error are shown.

ordered list of transformations is suggested. For many spreadsheets that are similar but not entirely uniform, the scripts might not generalise well and a lot of effort is required. An example of this case is discussed in Appendix 1.B.

FlashRelate learns an extraction program rather than a transformation program. In cases where many sparse columns have to be folded together, many examples might still be needed.

Foofah is closest to the approach presented here, as it synthesises transformation programs similar to ours. It requires complete input-output examples which might not be easy to specify for large tables.

6 Conclusion

In this paper we introduced the problem of automatically wrangling relational data from spreadsheets. We presented a simple heuristic greedy algorithm that is able to synthesise STPs for simple spreadsheets. Parameter pruning for the transformations in the transformation language is an important property of the algorithm. Real world spreadsheets can be correctly wrangled into a relational format, given a distinction in types is known.

In future work, we want to expand on the set of used transformations and the way their parameters are pruned. More specifically, we want to perform extended analysis of their completeness with respect to real world spreadsheets. For now, parameter pruning is largely based on intuition. Furthermore, we are looking at how the type spreadsheet can be automatically generated using hierarchical clustering.

7 Acknowledgements

This work is part of the ERC Advanced Grant SYNTH – Synthesising inductive data models, see `http://synth.cs.kuleuven.be`.

References

1. Data Wrangling Automation, IEEE International Conference on Data Mining (2016), `http://users.dsic.upv.es/~flip/DWA2016/`
2. Barik, T., Lubick, K., Smith, J., Slankas, J., Murphy-Hill, E.: Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets. In: Proceedings of the 12th Working Conference on Mining Software Repositories. pp. 486–489. IEEE Press (2015)
3. Barowy, D.W., Gulwani, S., Hart, T., Zorn, B.: Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In: ACM SIGPLAN Notices. vol. 50, pp. 218–228. ACM (2015)
4. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1), 10–18 (2009)
5. Harris, W.R., Gulwani, S.: Spreadsheet table transformations from examples. In: ACM SIGPLAN Notices. vol. 46, pp. 317–328. ACM (2011)
6. Jin, Z., Anderson, M.R., Cafarella, M., Jagadish, H.: Foofah: Transforming data by example. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 683–698. ACM (2017)
7. Kandel, S., Paepcke, A., Hellerstein, J., Heer, J.: Wrangler: Interactive visual specification of data transformation scripts. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 3363–3372. ACM (2011)
8. Polozov, O., Gulwani, S.: Flashmeta: A framework for inductive program synthesis. ACM SIGPLAN Notices 50(10), 107–126 (2015)
9. Raman, V., Hellerstein, J.M.: Potter's wheel: An interactive data cleaning system. In: VLDB. vol. 1, pp. 381–390 (2001)
10. Raza, M., Gulwani, S.: Automated data extraction using predictive program synthesis (January 2017), `https://www.microsoft.com/en-us/research/publication/automated-data-extraction-using-predictive-program-synthesis/`
11. The World Bank: World development indicators. `http://databank.worldbank.org/data/reports.aspx?source=world-development-indicators` (2016)

Appendix 1.A Transformations

Operation	**Explanation**	**Error**	**Arguments**
DELETE	Delete all rows which have an empty value in a given column c.	The number of non-empty deleted cells of rows that are *less specific* than a row that is kept. A row is less specific than another row if it contains less values and all of its non-empty values are equal.	$(1, \ldots, m)$
DROP	Delete column c.	Number of no-empty cells in the deleted column.	$(1, \ldots, m)$
FOLD	Fold x columns into two columns by replacing each row with x new rows where all other columns have the same value and the two new columns contain the old column name and value.	0	All sets of subsequent columns that have the same type.
FORWARDFILL	Fill each empty cell in a column with the first preceding non-empty value.	0	All columns with at least one missing value that do not have the first element missing.
SPLIT	Split a column on type.	0	All columns with at least two different types that are not $\varnothing$.
TRANSPOSE	Transpose rows and columns.	0	$\varnothing$

Table 1: Spreadsheet layout transformations, their reconstruction error and possible arguments.

Appendix 1.B Example Use Case: World Development Indicator Data

The World Bank provides an interface to download World Development Indicator data in spreadsheet format [11], some exports of which were also found in Fuse. Using default parameters and the *codes only* option, an example exported spreadsheet is shown in Figure 3a. We can also find a classification for income groups of countries – already in relational format – as shown in Figure 3b.

Suppose we want to learn a classifier for income groups using a relational learner such as Tilde, using the WDI data as features. A desired format for the WDI data could be a relation

`(country, indicator, year, value)`

which the learner can associate with the `(country, class)` relation to learn a classifier. Running our naïve synthesis algorithm on Figure 3a results in a simple STP

Fold(2, 6)
Delete(3)

that does the job. It managed to prune missing data as well. Other subsets of WDI data can be wrangled in the same way, not requiring any examples or intent.

Country Code	Series Code	2012	2013	2014	2015	2016
AFG	PA.NUS.PPP.05					
AFG	PA.NUS.PRVT.PP.05					
AFG	EG.CFT.ACCS.ZS	18.132	17.7192	17.3057		
DZA	PA.NUS.PPP.05					
DZA	PA.NUS.PRVT.PP.05					
DZA	EG.CFT.ACCS.ZS	99.99	99.99	99.99		

(a) A sample of World Development Indicator data from [11]. Types are indicated using grayscales colors.

AFG	Low income
ALB	Upper middle income
DZA	Upper middle income
ASM	Upper middle income
AND	High income
AGO	Lower middle income
ATG	High income

(b) Excerpt of an income group classification spreadsheet.

Fig. 3