Automatically Wrangling Spreadsheets into Machine Learning Data Formats

Gust Verbruggen[1] and Luc De Raedt[1]

KU Leuven, Belgium
{gust.verbruggen, luc.deraedt}@cs.kuleuven.be

Abstract. To help automate the important pre-processing step in machine learning and data mining, we introduce SYNTH-A-SIZER, a tool for semi-automatically wrangling spreadsheets into attribute-value format, so that they can be used by popular machine learning tools, only requiring the user to mark cells belonging to one single example. SYNTH-A-SIZER is based on inductive programming principles. We introduce SYNTH-A-SIZER's transformations, search algorithm as well as a heuristic and distance measure for identifying types. We also report on a first experimental evaluation.

Keywords: data wrangling, program synthesis, spreadsheets, preprocessing, inductive programming

1 Introduction

One long term goal of automatic machine learning and data science is to enable naive end-users to automatically analyse their data. Today we are far away from reaching that goal for two reasons. First, it is often hard to select the right learning setting, algorithm and parameters for the learning task. Second, it is well-known amongst data scientists that 80% of the time is spent on pre-processing and only 20% on the actual machine learning or data mining [5].

Looking at the state of the art in machine learning and data mining reveals that the first problem is receiving a lot of attention in the emerging area of automated machine learning [9]. Many impressive results have already been obtained and powerful tools have already been developed [7, 14]. Although there exist some tools that aid in the automatic preprocessing of data, especially with respect to feature construction [4], other preprocessing steps remain very challenging. Data wrangling is one of the most important ones.

This paper addresses exactly this issue and studies how to help end-users with data wrangling, that is, the process of transforming their data in the right format for data analysis. As non-experts often gather their data in spreadsheets, we focus on the question as to 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 [10] and KNIME [3]. Thus, we want to help fully automate the data wrangling process [1].

Several approaches for data wrangling with minimal user effort already exist. For example, the WRANGLER [12] system provides an interactive interface for creating transformation programs without needing to write code. Instantiations of the FLASHMETA [13] framework allow for synthesising data transformation programs by providing input-output examples. A notable instantiation is FLASHRELATE, [2] which extracts relational data from spreadsheets. More recently, FOOFAH [11] combine these two: transforming a spreadsheet based on examples.

In this paper, we take a next step in these developments and explore whether these processes can be automated while focussing on data in tabular form. A key difference with other approaches is that we focus on wrangling of machine learning data sets. While the above mentioned approaches use examples of the desired input-output behaviour to guide the program synthesis, we focus on the desired target *format* of the output. In this paper, the desired output is is in attribute-value format, which is used when working with tools such as WEKA. This format has distinct properties that we exploit in order to mediate the need for examples describing the desired output. Although we focus on the attribute-value format here, we believe that the principles and techniques we introduce could also be useful for relational learning.

This paper contributes a tool, SYNTH-A-SIZER, for semi-automatic data wrangling of machine learning datasets from minimal user input. SYNTH-A-SIZER uses a predictive program synthesis approach that transforms semi-structured data into a propositional format, for use in data analysis systems, from one positive example. The technical innovations of SYNTH-A-SIZER are that (1) we focus on the desired target format of the output and allow the user to provide hints about the target format using a new notion of *coloring*; (2) we introduce a domain specific language with an accompanying syntactic bias which allows to restrict the search space; and (3) we employ a novel type-based heuristic to assess and evaluate the transformations.

2 Motivating Example

Suppose a clothing store owner keeps two spreadsheets, containing sales and properties of clothing, respectively, an excerpt of which is shown in Table 1. The rightmost column in Table 1b is to be predicted. Given data in the correct format, plenty of tools are available to perform this task.

First, however, a user would need to know about transformations to unpivot and join tables together. In OpenRefine[1] an additional forward filling operation is required and Wrangler [12] make no assumptions about the output format, giving unpivot as the last suggestion. This motivates our belief that existing data wrangling tools are aimed at data scientists and other people who know their way around transforming data.

Our approach, on the other hand, is aimed at users who have no knowledge about transforming data at all. Currently, the only required interaction is *selecting*

[1] `www.openrefine.org`

Table 1: Spreadsheet data about clothing sales. Some cells are colored as they have been selected by the user.

(a) Sales data.

	29/08/2013	31/08/2013	09/02/2013
1006032852	2114	2274	2491
1212192089	151	275	570
1190380701	6	7	7
966005983	1005	1128	1326
876339541	996	1175	1304
1068332458	4	5	11

(b) Clothes' properties. On the right is the target, with one missing value.

1006032852	Low	4.6	Summer	o-neck	sleevless	**1**
1212192089	Low	0	Summer	o-neck	Petal	**0**
1190380701	High	0	Automn	o-neck	full	**0**
966005983	Average	4.6	Spring	o-neck	full	**1**
876339541	Low	4.5	Summer	o-neck	butterfly	**0**
1068332458	Low	0	Summer	v-neck	sleevless	

(c) Tables 1a and 1b wrangled into attribute-value format. This is what the user doesn't see, but what is generated by SYNTH-A-SIZER and used by data mining tools in the background in order to predict the last column.

1006032852	29/8/2013	2114	Low	4.6	Summer	o-neck	sleevless	1
1006032852	31/8/2013	2274	Low	4.6	Summer	o-neck	sleevless	1
1006032852	09/02/2013	2491	Low	4.6	Summer	o-neck	sleevless	1
1212192089	29/08/2013	151	Low	0	Summer	o-neck	Petal	0

⋮

1068332458	31/08/2013	5	Low	0	Summer	v-neck	sleevless	
1068332458	09/02/2013	11	Low	0	Summer	v-neck	sleevless	

values in the spreadsheet, as shown in Table 1. Wrangling is then performed in the background.

3 Problem Statement

The problem this paper wants to solve is best described on two levels. On a higher level, we aim to enable users without experience in programming or data wrangling to apply machine learning techniques to their data. As data gathered by such users is typically stored in a spreadsheet, we focus on the problem of mapping a spreadsheet S into a dataset D that can serve as the input to a machine learning algorithm. The machine learning algorithm should then generate a hypothesis h that can be applied to the dataset D to yield $h(D)$. Ideally, this approach allows for mapping $h(D)$ back into the spreadsheet S so that it can be shown to the user. The ultimate goal is that the transformations, the resulting dataset and the hypothesis are all constructed behind the scenes. Everything the user would see is the original spreadsheet S that has now been extended with the results of $h(D)$. A necessary condition for this to work is that the original spreadsheet S has been formatted in a *systematic* manner.

On a lower level, the problem we tackle in the present paper is to find the program f that maps $f(S) = D$, which is a program synthesis problem where a data wrangling program is learned.

3.1 Notation

As common in spreadsheets, the basic structure our programs transform are tables. A table is represented by an $m \times n$ matrix T. The element on column i and row j is referred to as $t_{i,j}$. We adopt a slicing notation $a : b$ to denote a range $(a, a+1, \dots, b)$ of cells, represented as a list of values. When a and/or b are omitted, the range extends to the size of the table, such that for example the values in row j are retrieved as $T_{:,j}$.

An m-ary relation $R \subseteq (A_1, \dots, A_m)$ of n tuples can be easily represented by a set of such tables. In the trivial case, every tuple becomes a row and each attribute is contained in a column of a single $m \times n$ table. This is the desired *target* data format for attribute-value learners such as those available in WEKA and KNIME. In the real world, however, the data can be spread out over multiple tables. Furthermore, values can be repositioned, empty cells and *spurious* cells can be added to the tables. The goal will then be to extract an equivalent table in the target format.

Example 1. Suppose we have a relation of car sales indicating whether a salesperson of a certain level gave a reduction or not:

$$\left\{\begin{matrix} \text{(Tim, junior, Audi, A1, no),} & \text{(Tim, junior, BMW, 1, yes),} \\ \text{(Megan, senior, Audi, A1, no),} & \text{(Megan, senior, Audi, A4, yes)} \end{matrix}\right\}$$

There are various ways of representing this relation in a set of tables, two examples of which are given in Figure 1. In the *Sales* table in Figure 1, some spurious values were added to denote the proportion of reductions given.

Tables can be transformed by transformations, which take as input one or more tables, optionally some arguments, and return a single table. The result of applying a transformation on some table(s) is then a new table with the elements from the original table(s) combined, repositioned, replicated or removed. We write $p = (\phi, \boldsymbol{a}) : \boldsymbol{T} \rightarrow T'$ for a table transformation p consisting of a transformation ϕ and a tuple of arguments $\boldsymbol{a}$, taking a set of tables $\boldsymbol{T}$ and returning a new table $T' = \phi(\boldsymbol{T}, \boldsymbol{a})$. We restrict ourselves to transformations that only change the layout of the spreadsheet, leaving cell values untouched. Each transformation ϕ has a set of valid arguments given a set of tables, denoted as $A_\phi(\boldsymbol{T})$.

Applying a transformation results in a *reconstruction error*, a measure of how much information is lost when it is applied to $\boldsymbol{T}$, written as $error(p, \boldsymbol{T})$. A transformation can be *inverted* if there exists a transformation $p^{-1} = (\phi^{-1}, \boldsymbol{a})$ such that $p^{-1}(p(\boldsymbol{T})) = \boldsymbol{T}$.

Example 2. Given a simple table

$T =$

Audi	A1
	A3
	A4

and the $\mathsf{Fill}(direction, i)$ transformation (see also Table 2), which fills empty values in column i with the value above ($forward$) or below ($backward$) it, we get $A_{\mathsf{Fill}}(T) = \{(forward, 1); (forward, 2); (backward, 1); (backward, 2)\}$.

A set of transformations $\mathcal{L}$ then serves as a simple domain-specific language (DSL) for wrangling tabular data. A table transformation program $\mathcal{P}$ is a sequence of transformations $(p_1, p_2, \ldots, p_k)$ with $p_i = (\phi_i, \boldsymbol{a}_i)$. Applying it to a table T is computed as $\mathcal{P}(\boldsymbol{T}) = \phi_p(\ldots \phi_2(\phi_1(\boldsymbol{T}, \boldsymbol{a}_1), \boldsymbol{a}_2) \ldots, \boldsymbol{a}_p)$. The definitions of reconstruction error and invertibility naturally extend from one transformation of a sequence of transformations.

3.2 Problem statement

We can now specify the program synthesis problem as follows.

Given *a set of tables $\boldsymbol{T} = (T_1, \ldots, T_k)$, a set of colorings C (cf. below), a scoring function $score(\boldsymbol{T}, C)$, and a set of transformations $\mathcal{L}$,* **find** *a transformation program P^* over $\mathcal{L}$ such that $P^* = \arg\max_P score(P(\boldsymbol{T}), C)$.*

The assumption is that there is an unknown target relation R and a program P^t such that $P^t(\boldsymbol{T})$ and R are equivalent (notation $P^t(\boldsymbol{T}) \equiv R$. The equivalence would account for row and column permutations. But the relation R is unknown and therefore we can only estimate how good any $P(\boldsymbol{T})$ is through a scoring function. This scoring function should recognise tables that are in attribute-value form. Such tables have rows that correspond to examples and columns that correspond to attributes. As a simple aid for recognising this, our scoring function can currently make use of one additional, user-provided input.

Essentially, the user is requested to color a set of cells that describe one example, possibly using different colors. The idea behind the coloring is twofold. First, cells belonging to one coloring should be mapped onto a single row. Cells in different tables with the same color, should be mapped onto the same cell in the target table. Second, all values in a single column should belong to the same attribute and should therefore be of the same type. If a colored cell occurs in a column, all other values in that column should be of the same type as the colored cell. Formally, a coloring C is a mapping from a set of cells $t_{i,j}$ to a set of colors. An example is given in Example 3.

While earlier work [15] assumed that the types were given, with each attribute having a different type, the present approach uses an edit-distance measure to determine how similar the type of two cells is. More specifically, it is assumed that the distance between different elements belonging to the same type is small—smaller than the distances between values of different types. The scoring function should then take into account (1) the quality of the rows and columns with respect to a coloring and (2) the reconstruction error.

Example 3. A cell coloring

$$C_1 = \{People_{1,1} \rightarrow \square, People_{2,1} \rightarrow \square, Sales_{1,1} \rightarrow ▨,$$
$$Sales_{1,2} \rightarrow \blacksquare, Sales_{2,2} \rightarrow \square, Sales_{3,2} \rightarrow \blacksquare\}$$

is shown on the left in Figure 1. After successfully wrangling it, these tables are transformed into the table on the right. Cells in each column are syntactically similar to a colored cell, no more empty values are present and the coloring contains an assignment that spans exactly one row. The transformed table should then get a much better score than the original ones.

People

Tim	Junior
Megan	Senior

Sales

Audi		
A1	Tim	no
A1	Megan	no
A4	Tim	yes
		2/3
BMW		
1	Megan	yes
		1/1

Tim	Junior	Audi	A1	no
Tim	Junior	Audi	A4	yes
Megan	Senior	Audi	A1	no
Megan	Senior	BMW	1	yes

Fig. 1: Two tabular representations of the relation in Example 1. **(left)** Spread out over two tables. The *Sales* table additionally contains empty cells and values not in the original relation. An example coloring is also shown. **(right)** Trivial representation as a single table.

4 Program Synthesis

We now introduce a predictive synthesis approach to synthesise the table transformation programs from just one example—a single tuple in the output relation that is colored by a user. Rather than assigning a score to the program itself, as in regular optimisation-based program synthesis [8], the output of the program is scored. A search over the space of transformation programs, optimising this score, is then used to find the program that correctly wrangles the input. In order to guide this search, we put a syntactic bias on the arguments of each transformation, which actually encodes a set of constraints on the possible arguments a transformation can take.

In the remainder of the section, we first introduce the supported transformations and their syntactic bias. Afterwards, we show how they are used to guide two search algorithms towards a solution optimising our scoring function. Finally, we provide the details of our scoring function.

4.1 Transformations and Syntactic Bias

The supported transformations is inspired on existing approaches [11, 12]. They have been chosen such that a wide variety of real world wrangling scenarios can be solved. In order to support multiple tables, a Join transformation is added. The full list is presented in Table 2.

Given a set of input tables $\boldsymbol{T}$ and a list of transformations, we can easily start recursively enumerating all transformation programs in search of one that optimises the heuristic. This is very unlikely to find a correct transformation program as the search space grows exponentially. To make the search over transformations tractable, a syntactic bias is placed on their arguments.

The intuition behind our syntactic bias is very similar to *witness functions* in FlashMeta [13], where they restrict the arguments of a function given the input–output examples. We reduce $A_\phi(T)$ based on the coloring and our knowledge of the heuristic. For example, the Fill transformation may only consider columns that have exactly one colored cell. A Delete is not allowed to remove colored

Table 2: Transformations supported by SYNTH-A-SIZER, their effect on an $m \times n$ table T and how the set of valid arguments is reduced given a coloring C. In the column on the right, i and j range over the columns of the tables they correspond to, d and fwd range over the boolean values.

Transformation	**Effect**	$A_\phi(T, C)$
Fill(T, i, fwd)	Fill each empty cell in $T_{i,:}$ with the first non-empty value above ($fwd = 1$) or below ($fwd = 0$) it.	All (i, fwd) such that $T_{i,:}$ contains empty values and exactly one colored cell from C is in $T_{i,:}$.
Delete(T, i)	Delete all rows j where $t_{i,j}$ is empty.	All (i) such that $T_{i,:}$ contains empty values and no cells $\in C$ are deleted.
Fold(T, i, j, h, d)	Fold $T_{i:j,:}$ into one ($h = 0$) or two ($h = 1$) new columns. If $h = 1$, elements from the first row are used as a description for values $T_{i:j,y\neq 0}$. If $d = 1$, rows with empty values in the folded column are deleted.	All (i, j, h, d) such that $T_{i:j,:}$ contains exactly column y with n colored cells and $h = (n > 1$ and $T_{y,0} \in C)$.
Join(T^1, T^2, i, j)	(outer) Join tables T^1 and T^2 on columns i and j respectively.	All (i, j) such that $T^1_{i,:} \subseteq T^2_{j,:}$ or vice versa.

cells. We write the reduced arguments of ϕ on T given C as $A_\phi(T, C)$. The full syntactic bias for each transformation is given in Table 2.

4.2 Synthesis Algorithm

Our synthesis algorithm then performs a beam search over the space of transformation programs. The beam is defined using the scoring function detailed in the next section. Two variations are implemented: depth-first (DFS) and breadth-first (BFS), consecutively aimed at being faster versus more robust.

A priority queue is used to implement the search. Let b be the beam width. In DFS, at every iteration of the synthesis loop: the best table so far is fetched, its reduced set of possible transformations is computed, the results are scored and the b best extended programs are added back to the queue. In BFS, *every* element is replaced by its top-b transformed tables from different transformations as long as at least one of those b tables is better than the current one.

4.3 Scoring Tables

Given a set of tables and a coloring, we want to estimate how close the set of tables is to being the unknown target relation. In an attribute-value formatted table, all columns describe one attribute and should thus contain values of the same data type. Every cell in the coloring should then belong to one of these column types. The actual types are unknown, however. We therefore estimate how similar the type of two values is using a syntactic distance function, which is detailed in the next section. It is used to define the scoring function.

Let there be c different colors. Some transformations, such as Fill, may propagate colors to other cells in the table. We then first select an assignment $\boldsymbol{a} = \{t_{i_1,j_1}, \ldots, t_{i_c,j_c}\}$ such that as many different columns as possible have a colored value. Next, for each cell $t_{i,j}$ in the assignment, the average distance between the cell and all other cells in its column

$$\mathit{avg\_color}(t_{i,j}) = \frac{1}{m} \sum_{y=1}^{m} d(t_{i,j}, t_{i,y}) \tag{1}$$

is computed, as well as the proportion of empty values in this column.

$$\mathit{empty}(t_{i,j}) = \frac{1}{m} \sum_{y=1}^{m} (t_{i,y} \equiv \varnothing) \tag{2}$$

These two values are added for each colored cell and then averaged over all colored cells in the selection to compute the final score.

$$\mathit{score\_color}(\boldsymbol{a}) = \frac{1}{c} \sum_{x=1}^{c} (\mathit{avg\_color}(t_{i_x,j_x}) + \mathit{empty}(t_{i_x,j_x})) \tag{3}$$

Finally, the same procedure is repeated for columns without colored cells, the difference being that the average similarity between any pair of values is computed such that (1) becomes

$$\mathit{avg}(i_x) = \frac{2}{m(m+1)} \sum_{y=1}^{m} \sum_{z=y}^{m} d(t_{i_x,y}, t_{i_x,z}). \tag{4}$$

for some column i_x. This allows for wrangling with partial colorings and also provides some robustness. Scores for both types of columns are added to compute the final score. When scoring multiple tables, their individual scores are summed.

$$\mathit{score}(T) = \mathit{score\_color}(\boldsymbol{a}) + \frac{1}{m-c} \sum_{i \notin i_1, \ldots, i_c} (\mathit{avg}(i) + \mathit{empty}(t_{i,0})) \tag{5}$$

4.4 Cell distance

At the heart of this method is the function that computes the similarity in type of cells. We propose a syntactic similarity function between cell values, treating them as a sequence of character classes.

This method is heavily inspired by the *string edit distance* between two cell values, with two differences: every character is represented by its character class and addition and deletion of elements between specific character classes can be made cheaper, or example, between lower- and uppercase letters

First, both strings are tokenised according to a set of disjoint character classes, such as digits, lower- and uppercase letters, delimiters (`-`, `/`,...) and currency symbols. Every token is weighted with the number of characters it consumed.

Next, the token sequences are globally aligned using a custom substitution matrix. The final distance is then computed as the distance between aligned tokens, weighted both by their weight and a distance matrix.

Let $(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ be the aligned tokenisations of two strings $\boldsymbol{a}$ and $\boldsymbol{b}$, $w(a_i)$ the weight of token a_i and $cost(t_1, t_2)$ the cost of a substitution between tokens t_1 and t_2. The distance between $\boldsymbol{a}$ and $\boldsymbol{b}$ is then computed as

$$d(\boldsymbol{a}, \boldsymbol{b}) = \sum_i cost(a_i, b_i) \frac{|w(a_i) - w(b_i)|}{w(a_i) + w(b_i) + 1}. \quad (6)$$

5 Evaluation

We propose a method for generating sythetic data that can be used for evaluating SYNTH-A-SIZER. The core idea is to generate messy data from a clean dataset. We start from a table and apply a number of subsequent random inverse transformations, creating a synthetic input dataset. The number of inverse transformations applied is called the *depth* of the synthetic dataset.

More specifically, we generate inverse programs P_t and associated messy tables $D' = P_t^{-1}(D)$ and then attempt to synthesize programs P that restore the original dataset as $P(D')$. The results are evaluated in terms of *recall* and *precision*, respectively the proportion of rows in D that is also in $P(D')$ and proportion of rows in $P(D')$ that is in D. The supported inverse transformations are explained in Table 3. Because some of the inverse transformations have side effects, i.e., a Fill reorders the rows, a few constraints need to be placed on the generated inverse programs in order to prevent total destruction of the data. Most notably, a table can only be reordered once. Further details of the generation process will be made available in a longer version of the paper.

Three datasets from the UCI repository [6] were selected, based on some simple requirements: not being too large, our implementation is not yet optimised to scale, and containing at least some categorical attributes in order to generate interesting inverse programs. They are the `Breast Cancer`, `Auto MPG` and `Computer Hardware` datasets[2].

5.1 Increasing depth

We start by assessing the basic wrangling capability of SYNTH-A-SIZER. The first row of each dataset is colored and sets of 100 programs of increasing depths are generated. For both algorithms, the average recall and precision are plotted in terms of the depth in Figure 2. Both mixed BFS and DFS achieve almost perfect reconstruction for lower depths in most cases. As depth increases, performance drops. We can take a closer look at the performance by plotting the distribution of the precisions, as done in Figure 3.

In our experiments, there are two main reasons why tables are not perfectly wrangled. First, complex Fold operations are not always correctly detected,

[2] `https://archive.ics.uci.edu/ml/datasets/`

Table 3: Inverse transformations supported by the data generator.

Transformation	**Inverse**	**Inverse arguments**
$\mathsf{Fill}(T, i, fwd)$	If it was not sorted before: sort T on column i. For every pair of consecutive equal values, set the top ($fwd = 0$) or bottom ($fwd = 1$) one to $\varnothing$.	All (i, fwd) such that $T_{i,:}$ doesn't contain empty values.
$\mathsf{Delete}(T, i)$	Repeat $\sim U(0, n-1)$ times: – Generate $\sim U(1, m/2)$ random strings – Add a row to T with the strings in random locations that are not i	All (i) such that $T_{i,:}$ does not contain an empty value.
$\mathsf{Fold}(T, i, j, h, d)$	Duplicate rows such that the elements outside of columns $i : i{+}h$ are repeated $j - i$ times. Expand values in folded column(s) in groups of $j-i+1$ consecutive rows into new columns.	All (i, j, h, d) such that (1) if $h = 0$: values outside of column i are replicated between at least $n/2$ rows or (2) if $h = 1$: values in column i are replicated at least $n/2$ times.
$\mathsf{Join}(T^1, T^2, i, j)$	Look for functional dependency $i \to Y$ between column i and columns Y such that $\|Y\| = j$. Split table by removing columns Y and building new table from columns (i, Y).	All (i, j) such that there exists a functional dependency $i \to Y$ and $\|Y\| = j$.

resulting in zero precision. This happens more often in datasets which have more similar attributes, such as `Breast Cancer` and `Hardware`. Second, Fill is sometimes applied in the wrong direction, resulting in precisions depending on the number of unique elements in the filled column.

As SYNTH-A-SIZER relies on a distance measure between types, it is sensitive to how syntactically similar different types are. An interesting question for further research is how to combine the similarity with background information about the underlying types, as well as alternative approaches for type detection.

5.2 Resilience to Coloring

We then ask the question how sensitive SYNTH-A-SIZER is to which cells are colored. All inverse programs of depths 3-5 from the previous experiments for which DFS achieved perfect results are computed 10 times with different rows colored. The precision distributions of wrangling those tables using DFS are shown Figure 4. Only for the `Breast Cancer` data are the obtained results considerably worse, probably due to similar features across columns. For both other datasets, SYNTH-A-SIZER seems robust enough to work for arbitrary colorings.

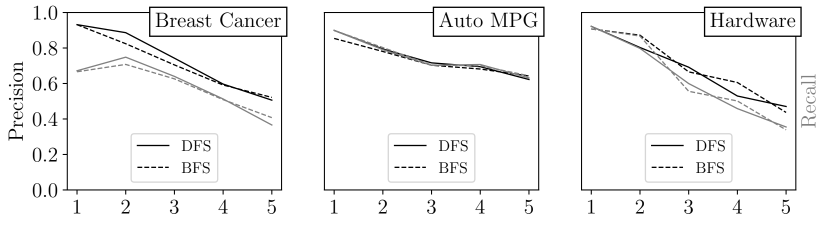

Fig. 2: Precision (black) and recall (gray) on three datasets for inverse programs of increasing depths (x-axis) using two synthesis algorithms. Due to good performance on the `Auto MPG` data, precision and recall are very similar.

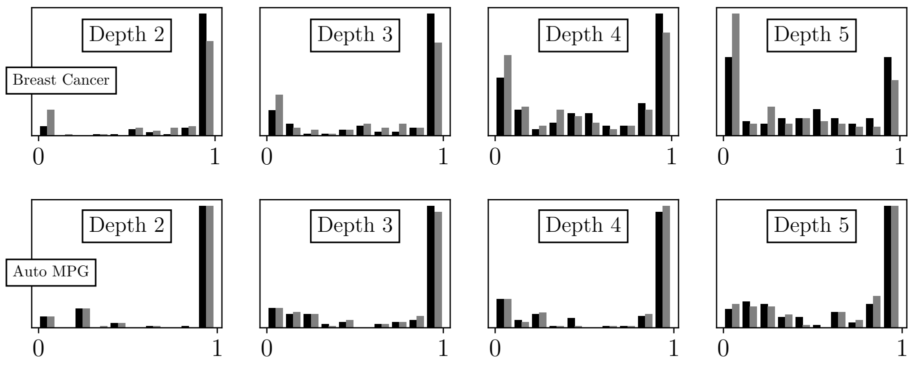

Fig. 3: Distribution of precision (black) and recall (gray) for increasing depths on the cancer (top) and auto (bottom) datasets.

6 Conclusion

We presented SYNTH-A-SIZER, a tool that semi-automatically wrangles attribute-value data from spreadsheets given only a coloring of one positive output example. Even though it uses a very simple heuristic and synthesis algorithm, it already achieves respectable performance on synthetically generated messy spreadsheets.

While more effort is required to improve the heuristic and distance function, these results provide a next step in the direction of fully automated wrangling of data from spreadsheets.

Acknowledgement This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No [694980] SYNTH: Synthesising Inductive Data Models).

References

1. Data Wrangling Automation, IEEE International Conference on Data Mining (2016), `http://users.dsic.upv.es/~flip/DWA2016/`

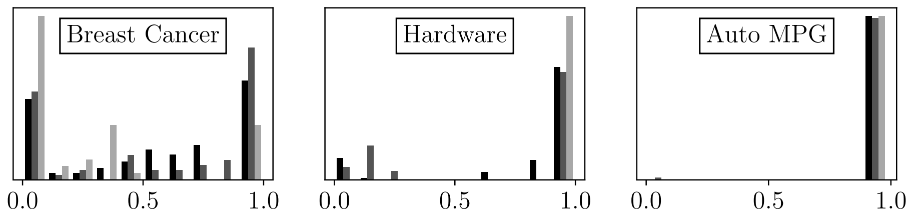

Fig. 4: Precision distributions of repeating previously successful runs with different colorings for all datasets.

2. 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)
3. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., Wiswedel, B.: Knime-the konstanz information miner: version 2.0 and beyond. AcM SIGKDD explorations Newsletter **11**(1), 26–31 (2009)
4. Boullé, M.: Towards automatic feature construction for supervised classification. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 181–196. Springer (2014)
5. Dasu, T., Johnson, T.: Exploratory data mining and data cleaning, vol. 479. John Wiley & Sons (2003)
6. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository (2017)
7. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems. pp. 2962–2970 (2015)
8. Gulwani, S., Polozov, O., Singh, R., et al.: Program synthesis. Foundations and Trends® in Programming Languages **4**(1-2), 1–119 (2017)
9. Guyon, I., Chaabane, I., Escalante, H.J., Escalera, S., Jajetic, D., Lloyd, J.R., Macià, N., Ray, B., Romaszko, L., Sebag, M., et al.: A brief review of the chalearn automl challenge: any-time any-dataset learning without human intervention. In: Workshop on Automatic Machine Learning. pp. 21–30 (2016)
10. 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)
11. 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)
12. 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)
13. Polozov, O., Gulwani, S.: Flashmeta: A framework for inductive program synthesis. In: ACM SIGPLAN Notices. vol. 50, pp. 107–126. ACM (2015)
14. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 847–855. ACM (2013)
15. Verbruggen, G., De Raedt, L.: Towards automated relational data wrangling. In: Proceedings of AutoML 2017@ ECML-PKDD: Automatic selection, configuration and composition of machine learning algorithms. pp. 18–26 (2017)