Domain specific induction for data wrangling automation (Demo)

Lidia Contreras-Ochando LICONOC@UPV.ES
Cèsar Ferri CFERRI@DSIC.UPV.ES
José Hernández-Orallo JORALLO@DSIC.UPV.ES
Fernando Martínez-Plumed FMARTINEZ@DSIC.UPV.ES
María José Ramírez-Quintana MRAMIREZ@DSIC.UPV.ES
DSIC, Universitat Politècnica de València, Spain

Susumu Katayama SKATA@CS.MIYAZAKI-U.AC.JP
University of Miyazaki, Japan

Abstract

Increasingly often, the available data we are analysing is messy, diverse, unstructured and incomplete, which makes its analysis more difficult. Thus, a data wrangling process is usually needed before most machine learning applications can be applied. This process aims at cleaning, transforming and combining data in order to render it in an appropriate format. Unfortunately, this process is mostly a manual and very time-consuming. In this paper we first present an approach to semi-automate some common transformations that appear in the data wrangling process, which is based on a general purpose inductive programming tool that is extended with domain-specific background knowledge. Next, we illustrate a web-based tool that allows users to provide a set of inputs and one or more examples of outputs, in such a way that a pattern is found that is applied to the rest of examples automatically by the tool.

Keywords: Data Wrangling Automation, Inductive Programming, Machine Learning

1. Introduction

In data science and machine learning applications, data wrangling includes tasks such as transforming data from one format to another with more structured and valuable form, cleaning datasets or combining them to create new attributes. In order to be more precise, we can think about datasets containing personal names with more than one surname, emails with different domains or dates in different formats, as the example on Table 1[1]. Normalising these data or extracting patterns from them can be highly complicated. Most of the times it is a manual, tedious, repetitive and boring process consuming up to 80% of the time in data science projects (Steinberg, 2013). There are tools (Gulwani, 2011; Ham, 2013; Kandel et al., 2011) or packages for different languages such as R (Boehmke, 2014; Wickham, 2016) or Python (Kazil and Jarmul, 2016) that facilitate the resolution of different problems related to data wrangling. Most of those data wrangling solutions are intended to be used by data analysts. However, in a data science project people with many different profiles and disciplines can take part and surely most of them have no programming skills. If we assume

1. Original dataset from: https://goo.gl/2BHoVS

that these transformations have to be solved only by programming, we are leaving out many people able to work with the data. Automating the data wrangling process is essential to reduce the time spent and the cost needed for the completion of data science projects, especially taking into account that the amount of available data is constantly increasing. Besides, making new intuitive and visual interfaces that are accessible to most of the people can be useful to increase the number of people working in data science projects.

Row No.	Borne sortie	Date sortie	Date retour	Borne retour
1	001	03/10/2016 00:18:36	03/10/2016 00:32:12	004
2	001	03/10/2016 00:25:45	03/10/2016 00:38:07	006
		...		
69852	001	6-10-16 20:35	6-10-16 14:39	097

Table 1: Subset of a real dataset of routes with a bike sharing system where dates are presented in two different formats. Borne/Date sortie represent the starting station and time where a user rents a bike. Borne/Date retour represent the station and time where the user returns that bike.

In this paper we present an approach to automate some transformation problems that appear in the data wrangling process, helping and making accessible the data cleaning and transformation steps by using inductive programming (Gulwani et al., 2015) and only few examples.

The paper is organised as follows. Section 2 describes our approach and presents the architecture of the demo tool. Section 3 shows results of several data wrangling problems solved using our approach. Finally, Section 4 concludes the paper and proposes ideas for future work.

2. Approach

This section briefly describes our approach. Instead of devising a new inductive, apropos, inductive engine, we re-use a general inductive programming system, MagicHaskeller, modifying it to take a configurable function library, which can be adapted for different data wrangling domains. Furthermore, and in order to illustrate how the approach works, we introduce a web-based application with an intuitive visual interface that enables users to solve several data wrangling problems for different data domains such as dates, emails, names and other string-based transformations.

2.1. Domain Specific Induction

Inductive programming (IP) has recently shown a high potential for automating the data wrangling process (Kitzelmann and Schmid, 2006; Gulwani et al., 2015). In contrast to other machine learning branches, inductive programming addresses the problem of learning small but complex programs from a very few representative examples generated when the user transforms one or more instances in the data. Successful applications include *FlashFill* (Gulwani, 2011), an Excel 2013 feature that automates repetitive string transformations using examples, and variants for other data manipulation problems, such as *FlashExtract* Le and Gulwani (2014) *FlashRelate* (Barowy et al., 2015) and *BlinkFill* (Singh, 2016)

However, these latter state-of-the-art IP systems are usually based on Domain Specific Languages (DSL), that is, languages specifically defined for a particular domain, where a

change of the DSL to cover other domains cannot be done by the user and might require a redesign of the system. Instead of this, and with the aim of increasing the range of applications, we propose the use of General Purpose Declarative Languages (GPDL) together with inductive programming tools.

Inductive programming with GPDL is normally inefficient due to the high search space required to find a solution. However, using a reduced set of functions for one specific domain (Domain-Specific Background Knowledge, DSBK), the search space can be limited by the number of functions. In this sense, our approach is based on *MagicHaskeller* (Katayama, 2012), a general-purpose inductive programming system for Haskell. MagicHaskeller learns from pairs of *input-output* examples, returning a list of functions f valid for converting $f(input)$ to *output*. Roughly speaking, in order to automate data wrangling problems we follow these steps:

1. We take a dataset of input-output pairs (Figure 1) or it is the user who, in the same way, can provide a few examples.
2. These examples are used as input predicates for MagicHaskeller.
3. MagicHaskeller returns the resulting function f.
4. The function f is applied to the rest of the inputs, obtaining the new values for the output column.

Although MagicHaskeller is very powerful, if we use the functions from the default library and the input example provided is complex, the induced solution might require the combination of many function symbols and MagicHaskeller may not find it. Aiming at overcoming this problem, we may be tempted to add many powerful and abstract functions to the library but, in this case, MagicHaskeller will have many primitives (functions) to choose from, and may not find the right primitive. In order to make MagicHaskeller scale up for a range of data wrangling problems, we have modified it to allow the system to have different domain-specific background knowledge. The integrated system is now able to switch the domain according to the problem. It should also be noticed that the *core* of MagicHaskeller had to be modified to allow the system to enable and disable the use of functions/primitives (for the different domains) at *runtime*, while keeping the general heuristics working. For this demo we have generated four different DSBKs to deal with four different types of data: dates, emails, names and strings in general. For each of these domains, the system has a different and specialised set of Haskell functions. For instance, related to the dates domain the tool has functions to change the punctuation mark within a date, extract the day or convert the month to numeric format. Despite the fact that we have taken into account only four domains for this version, our system can be easily extended to other domains since the background knowledge is based on a general purpose language (Haskell).

2.2. System Architecture and Functionality

As previously mentioned, in order to illustrate the main functionality of our approach, we have developed a basic web application (Figure 1). Even though our approach can deal with datasets including more than one input, in order to simplify the problem the main

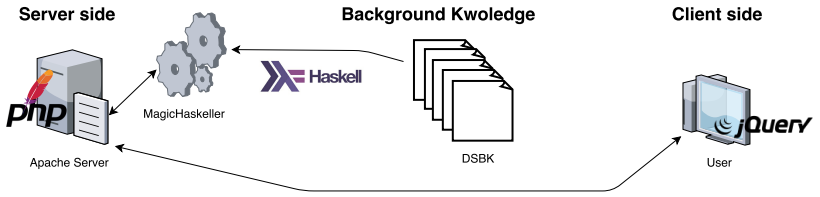

Figure 1: System architecture.PHP-based web application which works dynamically through AJAX requests (jQuery framework)

interface of this demo simulates a dataset table with text fields with only one input/output in each row. In this interface, the input field is used as a way to provide the original value for the attribute we want to transform (or clean), and the output field is the result we want to obtain. The input fields can be filled manually or automatically by means of using one of the example datasets provided by the tool. As it is illustrated in Figure 2, the goal of the system is, given just one (or very few) input/output example(s), to try to fill the outputs of the rest of instances (whose output fields have not been filled). For this, users can fill as many output text fields as they like (from 1 to $n-1$) in order to show the system what the output data should look like, i.e., which is the desired data transformation. Once the input/output examples are given, the user should choose one of the domains (DBBK) suggested by the system (dates, emails, names or strings). If no domain is selected, the system uses the general background knowledge (the complete set of functions with all the domains together) to try to induce the possible transformation to be applied.

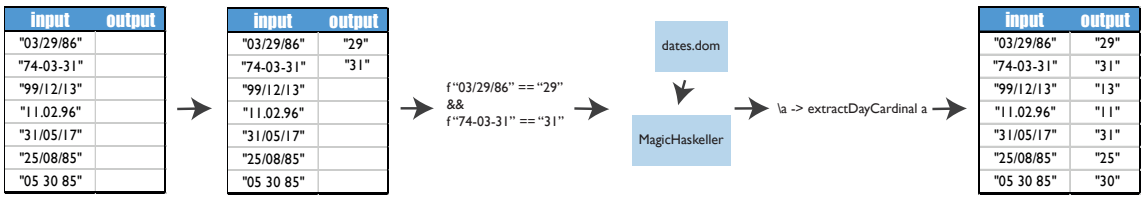

Figure 2: System functionality. Once the user provides input/output examples and the domain (DBBK), the system tries to induce the possible transformation to be applied.

MagicHaskeller returns, if possible, a list of functions solving the problem at hand. In this case, the system uses the first function of the list of potential functions returned (sorted by complexity). Then, Haskell is used to transform the rest of inputs provided in the website (with empty output fields) using the function selected. Once the process is completed, the empty output fields are updated in the website (Figure 3). This process is completely automatic and transparent for the user.

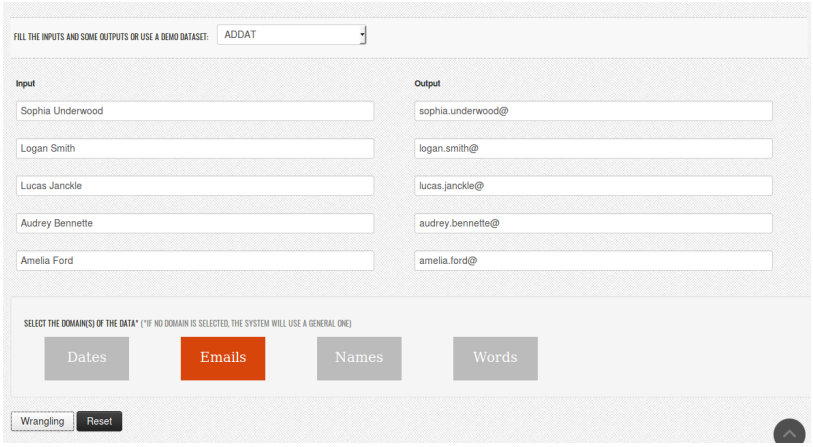

Figure 3: Outputs obtained using emails domain and the first function returned by MagicHaskeller: *toLowString (appendAt (changePunctuationString a dot))*. The first instance (row) is the input used to learn.

3. Experiments

In order to evaluate the performance of our approach, we have tested it with eight different datasets[2] originals or based on examples of Singh and Gulwani (2016), PROSE[3], Singh and Gulwani (2015), Gulwani (2011), Ellis and Gulwani (2017) and Contreras-Ochando et al. (2016). Table 2 illustrates some instances from these datasets including problems related to each different domain (dates, emails, names and strings).

In order to evaluate the performance of our approach we have compared our results with the results obtained using another data wrangling tool, FlashFill. FlashFill addresses a similar kind of problem, that is, it takes one o more examples to induce a potential solution to apply to the rest of instances. For these experiments, we have set the maximum number of functions concatenated by MagicHaskeller to four. However, this parameter could be higher, increasing the chances to get a solution. Table 2 shows some results and the accuracy obtained by using the two approaches with only the first example of the datasets. FlashFill works accurately with string-related problems (dataset #8, for instance). However, it has some limitations when the problem at hand includes data in different formats. Dataset #1 contains different types of dates, as the real example on Table 1. In this case, FlashFill is not able to detect that all the data are dates, transforming only a few of them. However, our approach is able to use functions adapted to the context, solving all of them regardless of their format. In summary, these results show that our approach is able to deal efficiently with data wrangling problems.

4. Conclusions and Future Work

In this paper we present an inductive approach to deal with data wrangling problems. Our approach is based on an existing inductive programming system, MagicHaskeller, but

2. The system has been tested in an Ubuntu 64-bit virtual machine (1 CPU, 2GB RAM).
3. https://github.com/Microsoft/prose/blob/master/ProgramSynthesis/benchmarks/emails.tsv

id	n	*input*	*output*	*FlashFill*	*DSI*
1	**7**	*29/03/86* 25-03-74 11.02.96	*29-03-86* 25-03-74 11-02-96	 25-03-74 11.02.96	 25-03-74 11-02-96
			Accuracy:	0.66	1
2	**7**	*03/29/86* 74-03-31 05 30 85	*29* 31 30	 03 30	 31 30
			Accuracy:	0.16	1
3	**5**	*Sophia Underwood* Logan Smith Lucas Janckle	*sophia.underwood@* logan.smith@ lucas.janckle@	 logan.smith@ lucas.janckle@	 logan.smith@ lucas.janckle@
			Accuracy:	1	1
4	**16**	*Nancy.FreeHafer@fourthcoffee.com* Andrew.Cencici@northwindtraders.com Laura.Giussani@adventure-works.com	*fourthcoffee* northwindtraders adventure-works	 northwindtraders adventure-works	 northwindtraders adventure-works
			Accuracy:	1	1
5	**4**	*Mr. Roger* Mrs. Simona Mr. John	*Male* Female Male		 Female Male
			Accuracy:	0	1
6	**5**	*Dr. Eran Yahav* Prof. Kathleen S. Fisher Bill Gates, Sr.	*Yahav, E.* Fisher, K. Gates, B.	 Sr., G.	 Fisher, K. Gates,, B.
			Accuracy:	0	0.5
7	**4**	*Dr. Mark Sipster* Louis Johnson, PhD Prof. Edward Davis	*Dr.* PhD Prof.	 Lou Prof.	 PhD Prof.
			Accuracy:	0.5	1
8	**3**	*International Business Machines* Principles Of Programming Languages International Conference on Software Engineering	*IBM* POPL ICSE	 POPL ICSE	 POPL ICSE
			Accuracy:	1	1

Table 2: Example of the results of our approach compared with *FlashFill*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn. Green colour means correct result; Red colour means incorrect result. n is the number of instances in each dataset.

modified and updated so that it allows the user to incorporate and use domain-specific background knowledge in runtime.

To illustrate its functionality, we have created a web application with an intuitive visual interface that facilitates the user to automate the data wrangling process. In order to evaluate the performance of the system we have tested it with four different types of data (domains) and problems: dates, emails, names and strings. Finally, we have compared our results with the results of another data wrangling system. The results show that our system is able to transform the input data into new output attributes from one or more examples in a few seconds.

As future work for the demo back-end, we propose to automate the detection of the data domain by using machine learning techniques to recommend the user the best DSBK given the input data provided. We also plan to extend the domains thus including new ones such as phone numbers, addresses, etc.

For the front-end of the web application, we plan to allow users to upload their own datasets in addition to the examples and the manual input. Besides, it could be useful to have more than one input to generate the output (for instance a user name and a domain to create an email).

Finally, we plan to use not only the first function returned by MagicHaskeller but some of them for making different dynamic suggestions to the user in order to decide which of them is the most suitable for the problem at hand.

Acknowledgments

This work has been partially supported by the EU (FEDER) and the Spanish MINECO project TIN 2015-69175-C4-1-R (LOBASS) and by Generalitat Valenciana under ref. PROMETEOII/2015/013 (SmartLogic). Lidia Contreras was supported by FPU-ME grant FPU15/03219.

References

D. W. Barowy, S. Gulwani, T. Hart, and B. G. Zorn. FlashRelate: extracting relational data from semi-structured spreadsheets using examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 218–228, 2015.

B Boehmke. Data processing with dplyr & tidyr. *Retrieved from RPub. com*, 2014.

L. Contreras-Ochando, F. Martínez-Plumed, C. Ferri, J. Hernández-Orallo, and M. J. Ramírez-Quintana. General-purpose inductive programming for data wrangling automation. *AI4DataSci @ NIPS 2016*, 2016.

K. Ellis and S. Gulwani. Learning to learn programs from examples: Going beyond program structure. May 2017. URL https://www.microsoft.com/en-us/research/publication/learning-learn-programs-examples-going-beyond-program/ /-structure/.

S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proc. 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

S. Gulwani, J. Hernandez-Orallo, E. Kitzelmann, S. H Muggleton, U. Schmid, and B. Zorn. Inductive programming meets the real world. *Communications of the ACM*, 58(11):90–99, 2015.

K. Ham. Openrefine (version 2.5). http://openrefine. org. free, open-source tool for cleaning and transforming data. *Journal of the Medical Library Association*, 101(3):233, 2013.

S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

S. Katayama. An analytical inductive functional programming system that avoids unintended programs. In *Proceedings of the ACM SIGPLAN 2012 workshop on Partial evaluation and program manipulation*, pages 43–52. ACM, 2012.

J. Kazil and K. Jarmul. *Data wrangling with Python: tips and tools to make your life easier*. O'Reilly Media, Inc., 2016.

E. Kitzelmann and U. Schmid. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research*, 7:429–454, 2006.

V. Le and S. Gulwani. FlashExtract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 542–553, New York, USA, 2014. ACM.

R. Singh. BlinkFill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment*, 9(10):816–827, 2016.

R. Singh and S. Gulwani. Predicting a correct program in programming by example. In *International Conference on Computer Aided Verification*, pages 398–414. Springer, 2015.

R. Singh and S. Gulwani. Transforming spreadsheet data types using examples. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 343–356, New York, USA, 2016. ACM.

D. Steinberg. How much time needs to be spent preparing data for analysis? *http://info.salford-systems.com/blog/bid/299181/How-Much-Time-Needs-to-be-Spent-Preparing-Data//-for-Analysis*, 2013.

H. Wickham. reshape2: Flexibly reshape data: A reboot of the reshape package. r package version 1.4. 2, 2016.