Automated Data Transformation with Inductive Programming and Dynamic Background Knowledge*

Lidia Contreras-Ochando[1]✉, Cèsar Ferri[1], José Hernández-Orallo[1], Fernando Martínez-Plumed[1], María José Ramírez-Quintana[1], and Susumu Katayama[2]

[1] Valencian Research Institute for Artificial Intelligence (vrAIn)
Universitat Politècnica de València, Spain
{liconoc,jorallo,mjramirez}@upv.es,{cferri,fmartinez}@dsic.upv.es,
[2] University of Miyazaki, Japan
skata@cs.miyazaki-u.ac.jp

Abstract. Data quality is essential for database integration, machine learning and data science in general. Despite the increasing number of tools for data preparation, the most tedious tasks of data wrangling –and feature manipulation in particular– still resist automation partly because the problem strongly depends on domain information. For instance, if the strings "17th of August of 2017" and "2017-08-17" are to be formatted into "08/17/2017" to be properly recognised by a data analytics tool, humans usually process this in two steps: (1) they recognise that this is about dates and (2) they apply conversions that are specific to the date domain. However, the mechanisms to manipulate dates are very different from those to manipulate addresses. This requires huge amounts of background knowledge, which usually becomes a bottleneck as the diversity of domains and formats increases. In this paper we help alleviate this problem by using inductive programming (IP) with a dynamic background knowledge (BK) fuelled by a machine learning meta-model that selects the domain, the primitives (or both) from several descriptive features of the data wrangling problem. We illustrate these new alternatives for the automation of data format transformation, which we evaluate on an integrated benchmark and code for data wrangling, which we share publicly for the community.

Keywords: Inductive Programming · Data Wrangling Automation · Declarative Programming Languages · Dynamic Background Knowledge

* This research was supported by the EU (FEDER) and the Spanish MINECO RTI2018-094403-B-C32, and the Generalitat Valenciana PROMETEO/2019/098. L. Contreras-Ochando was also supported by the Spanish MECD (FPU15/03219). J. Hernández-Orallo is also funded by FLI (RFP2-152). F. Martínez-Plumed was also supported by INCIBE (Ayudas para la excelencia de los equipos de investigación avanzada en ciberseguridad), the European Commission (JRC) HUMAINT project (CT-EX2018D335821-101), and UPV (Primeros Proyectos de lnvestigación PAID-06-18).

1 Introduction

Data science must integrate data from very different data sources (e.g., databases, repositories, webs, spreadsheets, documents, etc.). Rarely does this data come in a clean, consistent and well-structured way. Data wrangling, or data munging, is a process that usually involves data manipulation tasks that are repetitive, tedious and very time-consuming, such as transforming data into another format that can be properly processed and that makes the whole process more reliable [14]. Table 1 shows some data gathered in different formats, depending on the user's geographical region. Note that converting the (non-standardised) data from each column into a unified format needs a non-negligible manual effort.

Table 1: Example of personal data in different standard formats.

Name	**Email**	**Address**	**Phone**	**Date & Time**	**Count**
Alejandro Pala Corell	apalacorell@...	C/Jose Todos, 22	+34 465 698	03/04/17 19:39	Spain
Clau Bopper	clb@...	Rua bolog, 136	1195546	27/06/2017 22h56	France
Srta. Maria Garcia	mariagc@...	Av. Del Mar 14, piso 6, 12	659332	4 octubre 2017 10:20	Mexico
Dr Lauren Smith	Lauren.Smith@...	Flat 5, Royal Court, Coventy	748526	30 October 2017 9:45 am	UK
Sabrina Bunha Passa	sabrinabpassa@...	Rua Beni, 365, Alegre	+55 51 987	27/11/2017 07h05	Brasil
Mr David Bozz	David.bozz01@...	88 Lane of trees, Texas 77925	8259744	10/2/2018 12:30 PM	USA
Lara Alsi	lalsi@...	Av. Grande 2325 7p	54-12-3652	25/2/2018 17.00	Cuba

Recently, some tools have shown powerful skills in automating data wrangling tasks. Concretely, Inductive Programming (IP) [7,10] has been successfully applied to data wrangling problems. IP learns programs from a few examples possibly using declarative background knowledge (BK). From a machine learning (ML) point of view, the BK can be seen as a kind of bias, which is usually composed of a set of auxiliary primitives or concepts that can be combined to find a hypothesis that covers the data. But if this set of primitives becomes too large then the search for a suitable combination becomes huge. As usual, bias makes learning of some hypotheses easier (or possible) at the cost of other hypotheses. In general terms, *every problem becomes easy with appropriate background knowledge*. As a consequence, *the solution lies in finding this background knowledge*.

Consider Table 2. The difficulty of this problem lies in the different date formats, where the day can be the first, second or third number, and these numbers can be delimited by different symbols. A system based only on basic string transformations may never find the right solution using only one example since it does not know what the real problem is: extracting the first number? The first two digits? Or everything before any symbol? We must know how dates work, their constraints and how they are usually represented. We need BK.

In order to automate this process, the system (1) must recognise that it is handling names, dates or any other domain and (2) must have a sufficiently rich set of functions to deal with that particular domain. This size of the BK (the

Table 2: Example of a dataset with an input column composed of dates under very different formats and the output where the day of the month is extracted.

Id	Input	Output		Id	Input	Output
1	25-03-74	25		4	06 30 1975	30
2	03/29/86	29		5	25-08-95	25
3	1998/12/25	25		6	...	...

number of primitives) is known as *breadth* (b), while the minimum number of such primitives that have to be combined in the solution is known as *depth* (d). If we only provide very general primitives, d would increase considerably. However, as more kinds of domains are required, the library would become very large, and hence b. Clearly, both depth and breadth highly influence the hardness of the problem, jointly with the number of examples, n. Actually, for theory-driven induction, this hardness strongly depends on d and b, in a way that is usually exponential, $O(b^d)$ [12,6], with n being mostly irrelevant (indeed, most problems are solved from just one example). How can we keep both, and especially b, at very low levels?

In this paper, we propose to control the depth and breadth of the inductive inference problem by using *dynamic background knowledge* for each problem. We do this in three different ways. First, we structure the BK into specific subsets (domains) and select the most appropriate one. Second, we build a ranker that selects the most appropriate primitives depending on the problem. In both cases we use off-the-shelf ML techniques applied to a set of meta-characteristics based on the syntax of the inputs to be processed. Finally, we perform a combination of both approaches. As we will see, these approaches find a good trade-off between knowledge breadth and the solution depth. As a result, we solve effectively and efficiently a wide range of data wrangling manipulation problems, with the user just providing one example. For assessing the approaches, we introduce a new data wrangling benchmark consisting of a number of data transformation examples from the literature and others new problems.

This paper presents general ideas that go well beyond the particular use of IP to data science or other data manipulation applications. Overall, this paper contains four main contributions: (1) we show that the required breadth and depth for a particular theory-driven inductive inference problem can be minimised through the appropriate selection of primitives in the BK; (2) we propose several strategies to *dynamically* select or construct this appropriate BK automatically following the idea of detecting the best specialised functions according to the context of the particular problem to solve; (3) we develop and apply this schema to the important problem of data wrangling, which take a relevant portion of many data science applications; (4) we provide an open benchmark for further progress, replicability and comparison in this area. The paper is organised as follows. Section 2 summarises relevant related work. Section 3 addresses the problem of automating data wrangling with an IP system. Section 4 describes our approach for handling the BK. The experimental evaluation is included in

Section 5. Finally, Section 6 closes the paper with the conclusions and future work.

2 Related Work

Data wrangling is one scenario, among many others in data science and elsewhere, where learning must be done from very few examples. In these cases, the transfer or use of previous knowledge must impose a strong bias to make the problem solvable. In particular, the term 'inductive bias' refers to the assumptions a learning system does to prioritise some hypotheses over others [19]. In approaches where the hypothesis combines primitives or concepts, the inductive bias has the aim of adapting the depth –how many primitives or elements are needed– and breadth –how many choices there are in the librariy of components– of the learning process. Thus, with no alteration of the search procedure, the BK can be used to produce a bias on learning. However, as the BK grows to reduce the depth d for more and more problems, the search becomes intractable because of the growth of the breadth b. This problem has been analysed in incremental and lifelong learning scenarios [18,6,20]. The general idea is to combine the hypothesis generation process with a forgetting mechanism to limit the amount of BK that must take part in learning. The results shown in [25] suggest the usefulness of a measure of relevance on the BK to guide the search over programs relying on expert knowledge. In a recent work [17], the idea of ranking the functions according to some text features is presented. However, in this work the authors are based on the fact that input and output strings are related. For instance, the output is a substring of the input.

As said in the introduction, IP is an important paradigm in ML that is typically (but not always) theory-driven. IP is concerned with the problem of learning programs (typically recursive) from incomplete specifications such as input/output examples [10] and BK. The use of BK facilitates some problems but suffers from the general intractability issues when it gets large. Still, the great advantage of IP is that it can infer a solution for one or a few examples. In this regard, data wrangling and data transformation is one of the applications where IP has been shown very successful (see [26,4]), because the training data is generated online by the user, and we can only have a small number of examples (the benefit of automation disappears if the user has to provide many examples). IP has been so successful for data wrangling [10] that Microsoft included some of these tools in Excel, such as *FlashFill* [8]. One of the reasons of the success of these systems is the use of domain-specific languages (DSLs) [9], which are ad-hoc for data wrangling and data manipulation situations, and reduce the search space considerably.

With the goal of automatically transforming data within a spreadsheet format, Trifacta *Wrangler* [13] generates a ranked list of suggested transformations also inferred automatically from user input, the data type and some heuristics using programming-by-demonstration techniques. More recently, Neural Program Induction has been presented as an alternative for learning string transforma-

tions. In [21] the authors introduce a system that uses Neuro-Symbolic Program Synthesis to learn programs in a DSL by incrementally expanding partial programs. They perform experiments with I/O data wrangling examples having common substrings. Although the system is able to solve many of these examples it still has some limitations. On the one hand, the use of a DSL with many expressions implies a combinatorial explosion problem when a large number of programs have to be learned. On the other hand, due to their use of common substrings, the aforementioned example about date transformation, again, is impossible to solve. As another relevant work, in [22] the authors propose NPBE (Neural Programming by Example), a Programming by Example model based on deep neural networks. NPBE induces string manipulation programs based on simple I/O pairs by inferring the right functions and arguments. For assessing the validity of the induced programs, the authors create 1000 random examples using the *same* syntax structure, which is something that does not hold in general, as we see in Table 1.

Although the use of DSL systems for data wrangling automation seems prominent, it also brings further disadvantages: (1) using DSLs implies the use of languages that are specifically defined for a particular type of data processing. (2) Whenever a new application or domain is required, a new DSL has to be created, and the inductive engine recoded for it. (3) These systems work using a basic set of transformations, normally working with unique input-output pairs but not with an entirely table, and assuming the inputs of the same domain to be in a unique format. (4) DSL-based systems usually have 'program aliasing' problems (many different programs satisfying the examples) in such a way that more examples are needed to distinguishing the right hypothesis, affecting their performance [5].

Finally, the most recent work dealing with automatic data transformation is *TDE* (Transform Data by Example) [11]. In this work a search engine for Excel that indexes program snippets from different sources in order to find relevant functions for solving problems related to data transformation using two or more examples. Even when their results are better than other existing tools, the system uses more than 50k functions and their results tend to have many different solutions that the user has to select from. As we will show in section 5, this dependency on the user's manual effort results in worse results when the domain of the problem is not easy to detect.

3 Automating Data Wrangling with a general-purpose IP system

Instead of using DSLs for each particular context (e.g., dates, addresses, etc.), we propose to use a general-purpose IP system provided with a suitable domain (set of primitives) as BK. Hence, in our approach, the automation of data manipulation tasks is done as follows: (1) we take one example which is used to select the appropriate set of primitives that form the BK; (2) one or more examples are sent to an IP system, such as a first few rows in Table 2, which are the ones

a user could complete to trigger the process; (3) using the selected BK and the examples, the IP system learns a function `f` that correctly transforms the input of the examples to the given outputs; and (4) the function `f` is applied to the rest of the inputs, obtaining the new values for the output column automatically.

For the purpose of this work we have used *MagicHaskeller* [15] as the general-purpose IP system. The reasons for that choice are that *MagicHaskeller* exploits the power of the underlying language Haskell for very compact representations of the hypotheses, and it is able to solve many problems using only one example from the data. Despite this choice, the setting and the new techniques for dynamic knowledge allocation that we introduce in the next section could be replicated using other IP learning systems.

In a nutshell, *MagicHaskeller* receives an input example (`x`) and the expected result (`y`), and returns a list of functions (`f`) that make the values of the expressions `f x` and `y` be equal, which in Haskell notation is expressed as the Boolean predicate `f x == y`. *MagicHaskeller* looks for combinations of one or more functions (primitives) from its library to work like the `f` above. The solution (if exists) is a combination of d functions (where $d \leq d_{max}$). Trying to reduce d, we may be tempted to add a great number of powerful functions to the library. But, if so, *MagicHaskeller* will have many primitives to choose from (the breadth value b), suffering from a combinatorial explosion.

MagicHaskeller comes with 189 predefined primitives, the *default* BK, but they are insufficient for complex or very specific problems. In order to overcome this limitation, we have extended the generic BK including common string operators and nomenclature used in popular data manipulation tools (RapidMiner[1] and Trifacta[2]):

- **Constants:** Symbols, numbers, words or list of words.
- **Map:** Boolean functions for checking string structures.
- **Transform:** Functions that return the string transformed using one or more of the following operations:
  - **Add:** Appending elements to a string, adding them at the beginning, ending or a fixed position.
  - **Split:** Splitting the string into two or more strings by positions, constants or a given parameter.
  - **Concatenate:** Joining strings, elements of an array, constants or given parameters with or without adding other parameters or constants between them.
  - **Replace:** Changing one or more string elements by some other given element. This operation includes converting a string to uppercase and lowercase.
  - **Exchange:** Swapping elements inside strings.
  - **Delete/Drop/Reduce:** Deleting one or more string elements by some other given parameter, a position, size or mapping some parameter or constant.

[1] RapidMiner Studio - Feature List: `https://goo.gl/oYypMh`
[2] Trifacta Wrangler - Wrangle Language: `https://goo.gl/pJHSFw`

- **Extraction:** Get one or more string elements.

In total, we have added 108 functions, what we call the *freetext* BK. However, as we have seen with the example in Table 2, these functions may not be enough to solve specific tasks in some domains that would require more precise functions.

With this set of functions in the system's library, we are able to solve many common string manipulation problems. However, when data belong to a particular domain and the problem at hand ends up being a very exclusive task pertaining to that domain, more precise functions are needed in order to get correct results considering the context. We have explored the domains that are common in data wrangling problems (Excel[3] and Trifacta[4]) and we have created different Domain-Specific BKs (DSBK) according to them. We have modified *MagicHaskeller* so that a different DSBK can be used each time, as if it were selected by the user. All the DSBKs include specific functions for the domain and some *freetext* and *default* functions that can be useful for the specific problems of each domain as well. This is the final list of DSBKs [2]:

- *Dates* (23 domain-specific functions + 139 default/freetext functions): Extracting days from a substring, extending to a 4-digit full format, etc.
- *Emails* (23 domain-specific functions + 139 default/freetext functions): Getting all after the '@' symbol, append the '@' symbol, etc.
- *Names* (9 domain-specific functions + 93 default/freetext functions): Getting the initials of a name, creating a user login, etc.
- *Phones* (12 domain-specific functions + 104 default/freetext functions): Setting the prefix by country, detecting a phone in a text, etc.
- *Times* (5 domain-specific functions + 124 default/freetext functions): Change between 24/12h format, changing time zone, etc.
- *Units* (24 domain-specific functions + 124 default/freetext functions): Convert units of length, mass, time, temperature, etc.

We use the term *global* for the set of all primitives, including *default*, *freetext* and all the domain functions above mentioned (374 unique primitives). Of course, using this massive BK the system would not work, so one simple idea is to have the user choosing the appropriate domain in order to use the DSBK associated with the domain, an idea already explored in the literature to reduce the size of the BK [16]. However, in the long term, this is giving too much responsibility to the user. In the next section, we explore a new approach for automatically selecting a dynamic set of primitives for the BK.

4 Dynamic Background Knowledge

If we want to automatically detect the domain of a problem as humans do, we need a way to identify the characteristics that distinguish the domains. For instance, we can see that the '@' symbol is very distinctive for emails, while dates

[3] Excel - Data types in Data Models: `https://goo.gl/uWnbZh`
[4] Trifacta Documentation - Supported Data Types: `https://goo.gl/pV1owi`

in numeric format usually come with some specific punctuation for separating days, months and years. Following this idea, we have defined some descriptive *meta-features* that can be extracted automatically and describe different characteristics of the inputs, such as how the string starts (e.g., *start_upper*, *start_digit*, etc.), how it ends (i.e. *end_lower*, *end_digit*, etc.), which kind of symbols it contains (e.g., *has_numbers*, *has_dots*, etc.) and what structure they have (e.g., *is_onlyNumeric*, *is_onlyPunctuation*, etc.). We defined $n = 54$ meta-features in total, extracted by using regular expressions. Figure 1 shows an example of some of these characteristics extracted from dates and emails.

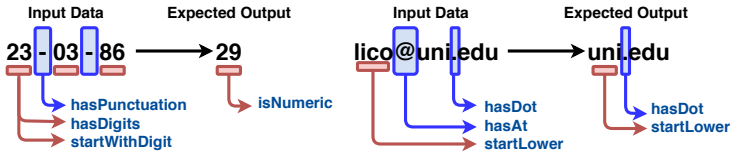

Fig. 1: An example of meta-features that can be extracted from the examples of different domains (dates and emails in the figure).

The idea of identifying domains was inspired by what a user would do to organise a large library of functions. But do we really need the notion of domain? Can we just do the selection of primitives by a ranking approach over the whole BK? As explained in the following paragraphs and illustrated in Figure 2, the information extracted from the input examples is going to be used in different ways:

1. **Domain identification for the appropriate DSBK** (*Inferred Domain).* As we want to automate the process, the domain can no longer be provided by the user, so we need to find a way to select the right domain for each problem. To do this, we train a *domain classifier* from a dataset composed of meta-features of m examples with correctly labelled domains. So, we have $n + 1$ columns (meta-features and domain) and m rows. The classifier is learned off-line with a pool of examples.
2. **Building dynamic BK by ranking the primitives from *global*** (*Ranking).* For this, we use the descriptive features for each example as input variables and the primitives that are used in the solution of the example as labels. We generate a *primitive estimator*, with the probability that a primitive may be needed for a particular problem. Since *global* has many primitives (374 primitives), we actually have a set of binary classifiers, one for each primitive, determining whether the primitives are required or not.
3. **Building dynamic BK by ranking the primitives from the identified domain** (*Inferred Domain +Ranking).* We also explore a combination of the two previous approaches. Namely, given a new problem, we first use the *domain classifier* to identify the most convenient domain according to the extracted features. Then we rank all primitives using the *primitive estimator*

but, in this case, only the functions included in the DSBK identified are taken into account. Finally, only the $k = 12$ best functions are used as BK[5].

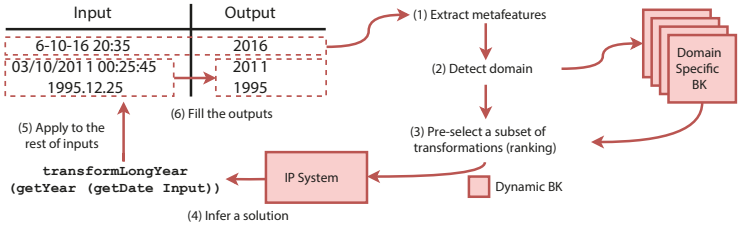

Fig. 2: Automating data wrangling with IP: process example. The first row (Input and Output) is used as an input example for the IP system. The function returned is applied to the rest of the instances to obtain the outputs.

5 Experiments

Unfortunately, at the beginning of this research there were no general benchmark or public repository to analyse the quality of new data wrangling tools [3]. In order to overcome this issue and for the experimental evaluation in this and future papers[6], we collected most of the examples previously presented in the literature [1,24,8,23]. In addition, we generated new examples based on the problems that appear in these papers. In total, we gathered 95 datasets (with 6 instances each) with different data wrangling problems including names, phones, emails, times and unit transformations. All the datasets are published as the first *data wrangling dataset repository*, openly available at `http://dmip.webs.upv.es/datawrangling/index.html` and are summarised in Table 3.

In this section, we present a summary of the results obtained by applying our system and other related systems on this repository. The complete results of these experiments can be found in [2], and the code is available at: `https://github.com/liconoc/DataWrangling-DSI`.

5.1 Strategies of employing BK functions

First, we want to determine which is the best strategy for selecting the BK to be used in data wrangling problems in such a way that the overall system is accurate and fast at the same time.

[5] We observed that the maximum number of functions needed to solve the most complex problem collected in our benchmark is $k = 12$.

[6] An application example of our system can be seen on: `https://www.youtube.com/watch?v=wxFhXYyonOw`

Table 3: Datasets included in the new data wrangling repository offered for the research community.

id	Description	Expected Output
1, 2	Add punctuation	The date in numeric format split by a punctuation sign
3 . . . 5	Change format	The date in one particular format
6, 7	Change Punctuation	The date in one particular format
8 . . . 10	Get Day	The day in numeric format
11, 12	Get Day Ordinal	The day in numeric ordinal format
13, 14	Get Month Name	The name of the month
15, 16	Get Week Day	The name of the weekday
17, 18	Reduce Month Name	The name of the month reduced to three letters
19, 20	Set Format	The date split in DMY format
21 . . . 23	Generate Email	An email account created with the name and the domain
24 . . . 27	Get After At	Everything after the at symbol
28, 29	Get Domain	The domain before the dot
30	Before At	Everything before the at symbol
31, 32	Add Title	The name with a title
33, 34	Get Title	The title attached to the name, if exists
35, 36	Generate Login	A login generated using the name
37 . . . 45	Reduce name	The name reduced before the surname(s)
46 . . . 50	Add Prefix by Country	Phone numbers with the prefix of the countries
51, 52	Delete Parentheses	The list of phone numbers without parentheses
53, 54	Get Number	A phone number presented in the string, if exists
55 . . . 59	Set Prefix	The list of phone numbers with the prefix
60, 61	Set Punctuation	A phone number split by a punctuation sign
62, 63	Add Time	The time increasing the hour by the integer
64, 65	Append o'clock Time	The time appending an o'clock time
66, 67	Append Time	The time appending the integer as new component
68, 69	Convert Time	The time formatted to 24 hours format
70, 71	Convert Time	The time formatted to a given format
72, 73	Convert Time	The time formatted to 12 hours format
74 . . . 77	Convert Time	The time changed from the first time zone to the second
78, 79	Delete Time	The time deleting the last component
80, 81	Get Hour	The hour component
82, 83	Get Minutes	The minutes component
84, 85	Get Time	A time presented in the string
86 . . . 89	Convert Units	The value transformed to a different magnitude
90, 91	Get System	The system represented by the magnitude
92, 93	Get Units	The units of the system
94, 95	Get Value	The numeric value without any magnitude

To build the *domain classifier* and the *primitive estimator*, we used the 54 descriptive meta-features and one off-the-shelf machine learning method: random forest (the learning method that obtained the best results [2]). We applied a leave-one-out cross validation approach using the 95 datasets, such that, for each fold, 94 datasets are used for training both classifiers and the remaining dataset is used for testing. As evaluation metrics we used accuracy for the domain

classifier, and *AUC* (the Area under the ROC curve which is a standard metric for ranking performance) for the primitive estimator. The results obtained for the *domain classifier* showed that the descriptive meta-features are useful to express the information about the domain since the classifier is able to predict the domain correctly 88.6% of the times (see Table 4). Analogously, the experiments performed with the *primitive estimator* (see [2] for details) obtained an average *AUC*=0.97, showing that it can predict accurately the functions needed to solve the problems.

Table 4: Results for the domain detection using the meta-features with different machine learning methods. The best results are highlighted in bold.

Method	**Acc.**	**Kappa**
C5.0 Tree	0.822	0.786
Neural Network	0.741	0.689
Naïve Bayes	0.458	0.350
Random Forest	**0.886**	**0.847**

The different strategies to configure the BK we experimentally analysed are:

1. *Default*: We use the default BK included in MagicHaskeller.
2. *Freetext*: We use the freetext BK (basic string transformation functions).
3. *Global*: We provide a BK composed by all the functions.
4. *User Domain*: We know (or the user gives) the correct domain (DSBK) for the problem.
5. *Inferred Domain*: We identify the domain of the problem automatically using the *domain classifier* and we select its associated DSBK.
6. *Ranking*: We rank all the functions of the global BK using the *primitive estimator*.
7. *Inferred Domain + Ranking*: We apply the *primitive estimator* to obtain the ranking of functions in the BK identified by the *domain classifier*.

We consider strategies 1, 2 and 3 as baselines since they do not constitute any improvement in the handling of the BK. Strategy 4 is included just as a human-assisted (semi-automated) reference, since it requires the manual recommendation of the appropriate DSBK. The experiments try to show whether our proposals (strategies 5, 6 and 7 introduced in Section 4) are able to improve the performance over the baselines, in time and accuracy. We also applied a leave-one-out cross validation using the 95 datasets, such that, for each fold from the six examples that contains the test dataset, only one random example is given to the IP system which, jointly with the *domain classifier* and the *primitive estimator*, infers a pattern that is applied to the five remaining examples. Accuracy is computed as the ratio of correctly covered examples by the induced pattern.

Table 5 shows the average time and accuracy for the seven strategies. The average times include the duration of the whole process: from the extraction of

Table 5: Average time (in seconds) and the average accuracy for the seven strategies. The best accuracy is highlighted in bold.

	Strategy	**time**	***acc***
1	Default	48.14 ± 28.46	0.09 ± 0.21
2	Freetext	78.77 ± 44.00	0.14 ± 0.25
3	Global	136.18 ± 67.97	0.06 ± 0.17
4	User Domain	74.23 ± 43.45	0.92 ± 0.20
5	Inferred Domain	75.45 ± 44.38	0.91 ± 0.24
6	Ranking	46.81 ± 25.51	**0.96 ± 0.12**
7	Inferred Domain + ranking	46.37 ± 26.89	0.94 ± 0.18

the first example to the automatic transformation of the rest of the outputs (as described in Figure 2). Concretely, we have measured: (1) time for detecting the domain (strategies 5 and 7); (2) time for ranking the functions (strategies 6 and 7); and (3) time of running the IP system (all strategies). In each execution we have used a $d_{max} = 12$ in *MagicHaskeller*, which means that the solution will have 12 functions at most (which is the number of functions selected by the ranking estimator as explained in Section 4). Considering the running times of Table 5, we conclude that the proposed strategies are able to speed up the whole process, especially those using the ranking of primitives.

If we consider accuracy, the baseline approaches are poor since they do not have the appropriate functions in the BK (*default* and *freetext* strategies), or there are too many functions to explore (*global* strategy). Strategies 6 and 7 are even able to outperform strategy 4, which requires a human. Only strategy 5 remains below this human-assisted reference. One of the reasons of these results is the misclassification of the emails domain, which means that strategy 5 is using an incorrect domain and the right solution is not obtained in this case. We can see this more clearly in Table 6, which shows the results by domain. Here we see that in some cases the baselines have too many functions and the system is not able to find the right solution. We can also see that the ranking of functions can achieve similar or better performance than the human-assisted reference.

5.2 Comparison with related systems

We have also compared the performance of our Dynamic BK selection approach using the ranking strategy with other data wrangling tools, specifically *FlashFill*, *Trifacta Wrangler* and *TDE (Transform Data by Example)*.

FlashFill works in a similar way as our approach, namely, it uses one, two or more input instances to try to infer a potential solution which is then applied to the rest of examples. *TDE* also works similarly except that it needs at least two instances for learning. However, *Trifacta Wrangler* works in a slightly different fashion: it tries to discover patterns and perform actions in the entire dataset. Each of these actions can involve one change (e.g., merge two columns) and they are saved in a final *recipe*. As we have used a d_{max} value equal to 12 in MagicHaskeller, in order to make a fair comparison with *Trifacta Wrangler*, we limit

Table 6: We show for the seven strategies, the average and standard deviation of time (in seconds), and the average and standard deviation of the accuracy depending on the domain. The best accuracy and time are highlighted.

domain	strategy	time	acc	domain	strategy	time	acc
dates	default	59.13 ± 24.22	0.15 ± 0.33	phones	default	34.93 ± 30.63	0 ± 0
	freetext	112.81 ± 41.15	0.34 ± 0.42		freetext	56.32 ± 45.23	0.24 ± 0.43
	global	188.36 ± 42.91	0 ± 0		global	95.43 ± 72.67	0.22 ± 0.43
	user_domain	114.27 ± 42.02	0.90 ± 0.29		user_domain	53.21 ± 47.49	**0.89 ± 0.32**
	pred_domain	117.86 ± 42.80	0.90 ± 0.29		pred_domain	51.04 ± 49.73	**0.89 ± 0.32**
	ranking	58.95 ± 24.42	**0.91 ± 0.29**		ranking	31.65 ± 28.80	**0.89 ± 0.32**
	dom_ranking	**58.36 ± 23.46**	**0.91 ± 0.29**		dom_ranking	**31.37 ± 27.95**	**0.89 ± 0.32**
emails	default	56.17 ± 31.81	0.08 ± 0.25	times	default	46.59 ± 26.41	0.10 ± 0.26
	freetext	81.17 ± 43.24	0 ± 0		freetext	75.21 ± 44.44	0.11 ± 0.28
	global	155.37 ± 77.75	0.08 ± 0.25		global	119.25 ± 71.74	0.020 ± 0.07
	user_domain	69.67 ± 37.21	0.98 ± 0.06		user_domain	78.46 ± 48.83	0.89 ± 0.22
	pred_domain	63.07 ± 34.99	0.90 ± 0.32		pred_domain	83.09 ± 51.50	0.89 ± 0.22
	ranking	56.17 ± 31.22	**1 ± 0**		ranking	43.92 ± 23.75	**0.98 ± 0.06**
	dom_ranking	**54.87 ± 29.45**	0.90 ± 0.32		dom_ranking	**43.79 ± 23.44**	**0.98 ± 0.06**
names	default	**49.95 ± 28.11**	0.04 ± 0.15	units	default	42.05 ± 29.59	0.16 ± 0.26
	freetext	82.35 ± 44.10	0.07 ± 0.26		freetext	64.75 ± 45.83	0.06 ± 0.13
	global	147.01 ± 69.02	0.07 ± 0.26		global	111.65 ± 73.74	0 ± 0
	user_domain	59.61 ± 33.38	**0.97 ± 0.07**		user_domain	70.15 ± 51.76	0.92 ± 0.25
	pred_domain	57.28 ± 34.80	**0.97 ± 0.07**		pred_domain	74.35 ± 52.43	0.92 ± 0.25
	ranking	**49.95 ± 28.54**	**0.97 ± 0.07**		ranking	40.25 ± 28.33	**1 ± 0**
	dom_ranking	**49.95 ± 28.54**	**0.97 ± 0.07**		dom_ranking	**39.85 ± 28.51**	**1 ± 0**

the maximum number of actions in each Wrangler recipe to 12. Additionally, although some tools are able to generate more than one solution, if they exist (as *TDE* and *MagicHaskeller* do), for the experiments we have only considered the first solution offered by the systems.

Due to space limitations, Table 7 shows some illustrative outcomes obtained by the analysed systems for some datasets (one dataset per each domain described in section 3) as well as their accuracy values. The first instance (in italics) for each dataset (*input* column) is the one used for inferring the solution (except for *TDE* that, as mentioned above, needs the two first instances for learning). The complete results of this comparison between systems can be found in [2].

Flashfill works well with emails and some basic string transformations, but it fails when it has to deal with people's names, problems related to phones or times, and dates in different formats. Something similar is observed in the *TDE* results: inconsistent data formats cause *TDE* finds incorrect solutions because it is not able to detect the domain or the problem at hand. On the other hand, *Trifacta Wrangler* is able to detect some data types or domains, for instance: 'url', 'time', 'phone' since it has some predefined formats for each domain. In this way the tool is capable of solving very domain-specific problems (e.g., getting

the month or the day in a date, detect an email or extract the hour of a time stamp), although with some limitations (e.g., it cannot deal with inconsistent or different formats in the same set of input data). The last problem of *Trifacta Wrangler* is that the user needs to know the language behind the tool or some regular expressions in order to solve more complex examples. On the contrary our system is able to solve most of the problems using only one example given by the user in the same way one can fill data in a spreadsheet, having into account that the user does not need to know any technical knowledge related to the system or the language behind it.

The authors of *TDE* have also created a benchmark of stackoverflow-related questions[7] that can also be used in order to test data transformation systems. We have tested our system with the 225 datasets of this benchmark in the same conditions as our system, i.e., using the first instance of each dataset as the input example for our system. In this way, our system solves 35.1% of these datasets, using the functions that we have defined. We have to consider that this benchmarck includes domains not defined in our system and some specific problems that need ad-hoc functions in order to be solved. Having in mind the examples not solved, we can include new functions in our system, for instance, new unit conversions or the extraction of plain text from languages such as HTML.

Finally, we can also compare our system with the Neuro-Symbolic Program Synthesis system of [21], at least conceptually, as it cannot be applied directly to the data wrangling repository. As we already discussed in the related work section, Parisotto et al. describe some problems that their system is not able to solve since they require four or more *Concat* operations. One of these problems is transforming phone numbers into a consistent format. For instance, given the input "(425) 221 6767" the expected output would be "425-221-6767", and given the input "206.225.1298" the expected output would be "206-225-1298". In this case, our system is able to solve this problem by using three basic primitives of the *freetext* domain. Besides this example, our system is able to solve some other examples that this kind of system does not solve since input and output have nothing in common. For instance, given the input "2pm" the expected output would be "14:00". This example implies knowledge of times and, in this case, our system is also able to solve the problem.

The comparisons above may look non-systematic, but all these approaches use different settings and additional data, apart from a very different number of examples, which makes the results not really comparable. This is one of the reasons why the presented benchmark and the minimum requirements of our method can be set as a baseline to beat by future variants of these and other approaches.

[7] TDE Benchmark: `https://github.com/Yeye-He/Transform-Data-by-Example`

6 Conclusions

Most data science applications require the manipulation of data that is in different formats. One key issue that humans rely on is their domain knowledge, which allows them to use primitives that are specific to the domain, when coding transformations. However, if a large number of primitives is included in the background knowledge to cover a variety of situations we get an intractable problem, as we have too many to choose from. In this paper, we have proposed different strategies that try to reduce the size of the background knowledge, based on an automated selection of the domain and/or a ranking of primitives to build the BK dynamically for each example. We have illustrated all this in the real problem of formatting data of very different domains from just one example.

To properly evaluate our system (and other existing and future data wrangling systems), We have introduced a new repository of 95 data wrangling datasets, which we make available for the community. We have performed experiments over this benchmark to illustrate the several strategies to the dynamic selection or construction of background knowledge, showing that they greatly improve accuracy and reduce time, especially strategy 6, the ranking approach.

Summing up, we have presented a data wrangling system that (1) uses off-the-shelf (and open) IP and ML techniques, (2) learns from one example, (3) is automated and does not require the user's input for the domain selection, and (4) covers a wide range of string manipulation problems, with results well above other approaches.

As future work, we plan to study the proposed strategies for other IP systems and domains to improve the system. We also want to consider other ways to solve the ranking of functions to avoid a fixed value of k, for instance, using a threshold based on the probabilities returned by the *primitive estimator*.

References

1. Bhupatiraju, S., Singh, R., Mohamed, A.r., Kohli, P.: Deep API programmer: Learning to program with APIs. arXiv preprint arXiv:1704.04327 (2017)
2. Contreras-Ochando, L.: DataWrangling-DSI: BETA - Extended Results (2019). https://doi.org/10.5281/zenodo.2557385
3. Contreras-Ochando, L., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., Ramírez-Quintana, M.J., Katayama, S.: General-purpose declarative inductive programming with domain-specific background knowledge for data wrangling automation. arXiv preprint arXiv:1809.10054 (2018)
4. Cropper, A., Tamaddoni, A., Muggleton, S.H.: Meta-interpretive learning of data transformation programs. In: Inductive Logic Programming. pp. 46–59 (2015)
5. Devlin, J., Bunel, R.R., Singh, R., Hausknecht, M., Kohli, P.: Neural program meta-induction. In: NIPS. pp. 2077–2085 (2017)
6. Ferri-Ramírez, C., Hernández-Orallo, J., Ramírez-Quintana, M.J.: Incremental learning of functional logic programs. In: FLOPS. pp. 233–247. Springer (2001)
7. Flener, P., Schmid, U.: An introduction to inductive programming. Artificial Intelligence Review **29**(1), 45–62 (2008)

8. Gulwani, S.: Automating string processing in spreadsheets using input-output examples. In: Procs. 38th Principles of Programming Languages. pp. 317–330 (2011)
9. Gulwani, S., Harris, W.R., Singh, R.: Spreadsheet data manipulation using examples. Communications of the ACM **55**(8), 97–105 (2012)
10. Gulwani, S., Hernandez-Orallo, J., Kitzelmann, E., Muggleton, S.H., Schmid, U., Zorn, B.: Inductive programming meets the real world. Communications of the ACM **58**(11), 90–99 (2015)
11. He, Y., Chu, X., Ganjam, K., Zheng, Y., Narasayya, V., Chaudhuri, S.: Transform-data-by-example (tde): an extensible search engine for data transformations. Proceedings of the VLDB Endowment **11**(10), 1165–1177 (2018)
12. Henderson, R.: Incremental learning in inductive programming. In: Int. WS on Approaches and Applications of Inductive Programming. pp. 74–92. Springer (2009)
13. 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)
14. Kandel, S., Heer, J., Plaisant, C., Kennedy, J., van Ham, F., Riche, N.H., Weaver, C., Lee, B., Brodbeck, D., Buono, P.: Research directions in data wrangling: Visualizations and transformations for usable and credible data. Inf. Visualization **10**(4), 271–288 (2011)
15. Katayama, S.: An analytical inductive functional programming system that avoids unintended programs. In: Procs. PEPM. pp. 43–52. ACM (2012)
16. Kietz, J.U., Wrobel, S.: Controlling the complexity of learning in logic through syntactic and task-oriented models. In: Inductive logic programming. Citeseer (1992)
17. Menon, A., Tamuz, O., Gulwani, S., Lampson, B., Kalai, A.: A machine learning framework for programming by example. In: ICML. pp. 187–195 (2013)
18. Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., et al.: Never-ending learning. Communications of the ACM **61**(5), 103–115 (2018)
19. Mitchell, T.M.: The need for biases in learning generalizations. Rutgers Univ. New Jersey (1980)
20. Mitchell, T.M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., Schlimmer, J.C.: Theo: A framework for self-improving systems. Architectures for intelligence pp. 323–355 (1991)
21. Parisotto, E., Mohamed, A.r., Singh, R., Li, L., Zhou, D., Kohli, P.: Neuro-symbolic program synthesis. arXiv preprint arXiv:1611.01855 (2016)
22. Shu, C., Zhang, H.: Neural programming by example. In: AAAI. pp. 1539–1545 (2017)
23. Singh, R., Gulwani, S.: Predicting a correct program in programming by example. In: Int. Conf. Computer Aided Verification. pp. 398–414. Springer (2015)
24. Singh, R., Gulwani, S.: Transforming spreadsheet data types using examples. In: Procs. 43rd Principles of Programming Languages. pp. 343–356 (2016)
25. Srinivasan, A., King, R.D., Bain, M.E.: An empirical study of the use of relevance information in inductive logic programming. JMLR **4**(Jul), 369–383 (2003)
26. Wu, B., Szekely, P., Knoblock, C.A.: Learning data transformation rules through examples: Preliminary results. In: Information Integration on the Web. p. 8 (2012)

Table 7: Results obtained by *FlashFill*, *Trifacta Wrangler*, *TDE* and our approach (Dynamic BK with ranking strategy), on a sample of datasets of six different domains. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill*, *MagicHaskeller* and *Trifacta Wrangler* to generate the solution. For *TDE* the two first examples are used. Green colour means correct result; Red colour means incorrect result.

id	*input*	*expected − output*	*FlashFill*	*Trifacta Wrangler*
	03/29/86	*29*		
8	74-03-31	31	03	03
	99/12/13	13	12	12
	11.02.96	11	02	
	31/05/17	31	05	05
	25-08-85	25	08	08
		Accuracy: 0		0
	Nancy.FreeHafer@fourthcoffee.com	*fourthcoffee.com*		
24	Andrew.Cencici@north-trad.com	north-trad.com	north-trad.com	north-trad.com
	Jan.Kotas@litwareinc.com	litwareinc.com	litwareinc.com	litwareinc.com
	Mariya.Sergienko@graphics.com	graphics.com	graphics.com	graphics.com
	Steven.Thorpe@northwindtraders.com	northwindtraders.com	northwindtraders.com	northwindtraders.com
	Michael.Neipper@northwindtraders.com	northwindtraders.com	northwindtraders.com	northwindtraders.com
		Accuracy: 1		1
	Dr. Eran Yahav	*Yahav, E.*		
37	Prof. Kathleen S. Fisher	Fisher, K.	Fisher, Kathleen S.	S, K.
	Bill Gates, Sr.	Gates, B.	Sr., G.	Sr, G.
	George Ciprian Necula	Necula, G.	Necula, C.	Necula, C.
	Ken McMillan, II	McMillan, K.	II, M.	II, M.
	Mr. David Jones	Jones, D.	Jones, D.	Jones, D.
		Accuracy: 0.2		0.2
	John DOE 3 . . . [TS]865-000-0000 . . .	*865-000-0000*		
53	A GEDA-. . . 865-001-0020 - - 5941-00 . . .	865-001-0020	865-001-0020	865-001-0020
	The quick, . . . 425-437-9620 69 11 60 20	425-437-9620	437-9620 69	425-437-9620
	425-457-2130, DJs flock by . . . : 18:95	425-457-2130	457-2130, DJs flock . . . : 18	425-457-2130
	425-618-4390 - 78 2642	425-618-4390	618-4390	425-618-4390
	17:58-19:29, 425-743-1650	425-743-1650	58-19:29	425-743-1650
		Accuracy: 0.2		1
	1:34:00 PM CST	*1:34:00*		
84	01:55	01:55	01:55	01:55
	3:40 AM	3:40	3:40	3:40
	07:05:59	07:05:59	07:05:59	07:05:59
	08:40 UTC	08:40	08:40	08:40
	16:15:12	16:15:12	16:15:12	16:15:12
		Accuracy: 1		1
	1441.8mg → g	*1.4418001*		
86	84kg → *g*	84000.0	8.4418001	
	14300ms → *s*	8700000.0	1.4418001	
	87 s → *ns*	8700000.0	8.4418001	
	12.20dg → *mg*	1220.0	1.4418001	
	1854 dam → *dm*	185400.0	1.4418001	
		Accuracy: 0		0