Towards Data Wrangling Automation through Dynamically-Selected Background Knowledge

Lidia Contreras Ochando

Supervised by
José Hernández Orallo
César Ferri Ramírez

Valencia, December 2020

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science

Towards Data Wrangling Automation through Dynamically-Selected Background Knowledge

By

Lidia Contreras Ochando

Supervised by

José Hernández Orallo	Universitat Politècnica de València
Cèsar Ferri Ramírez	Universitat Politècnica de València

Evaluation committee

Chair:	María Alpuente Frasnedo	Universitat Politècnica de València
Vocal:	Sebastijan Dumančić	KU Leuven
Secretary:	Fabrizio Riguzzi	Università di Ferrara

External reviewers

Andrew Cropper	University of Oxford
Sebastijan Dumančić	KU Leuven
Raúl Santos Rodríguez	University Of Bristol

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A thesis submitted for the degree of
Doctor of Philosophy in Computer Science

Departamento de Sistemas Informáticos y Computación

- Valencia, December 2020 -

This research was supported by the Spanish MECD Grant FPU15/03219; and partially by the Spanish MINECO TIN2015-69175-C4-1-R (Lobass) and RTI2018-094403-B-C32-AR (FreeTech) in Spain; and by the ERC Advanced *Grant Synthesising Inductive Data Models* (Synth) in Belgium.

To Francisco and Carmen — my parents.

Abstract

Data science is essential for the extraction of value from data. However, the most tedious part of the process, data wrangling, implies a range of mostly manual formatting, identification and cleansing manipulations. Data wrangling still resists automation partly because the problem strongly depends on domain information, which becomes a bottleneck for state-of-the-art systems as the diversity of domains, formats and structures of the data increases.

In this thesis we focus on generating algorithms that take advantage of the domain knowledge for the automation of parts of the data wrangling process. We illustrate the way in which general program induction techniques, instead of domain-specific languages, can be applied flexibly to problems where knowledge is important, through the dynamic use of domain-specific knowledge. More generally, we argue that a combination of knowledge-based and dynamic learning approaches leads to successful solutions. We propose several strategies to automatically select or construct the appropriate background knowledge for several data wrangling scenarios. The key idea is based on choosing the best specialised background primitives according to the context of the particular problem to solve.

We address two scenarios. In the first one, we handle personal data (names, dates, telephone numbers, etc.) that are presented in very different string formats and have to be transformed into a unified format. The problem is how to build a compositional transformation from a large set of primitives in the domain (e.g., handling months, years, days of the week, etc.). We develop a system (BK-ADAPT) that guides the search through the background knowledge by extracting several meta-features from the examples characterising the column domain. In the second scenario, we face the transformation of data matrices in generic programming languages such as R, using an input matrix and some cells of the output matrix as examples. We also develop a system guided by a tree-based search (AUTOMAT[R]IX) that uses several constraints, prior primitive probabilities and textual hints to efficiently learn the transformations.

With these systems, we show that the combination of inductive programming with the dynamic selection of the appropriate primitives from the background knowledge is able to improve the results of other state-of-the-art —and more specific— data wrangling approaches.

Resumen

El proceso de ciencia de datos es esencial para extraer valor de los datos. Sin embargo, la parte más tediosa del proceso, la preparación de los datos, implica una serie de formateos, limpieza e identificación de problemas que principalmente son tareas manuales. La preparación de datos todavía se resiste a la automatización en parte porque el problema depende en gran medida de la información del dominio, que se convierte en un cuello de botella para los sistemas de última generación a medida que aumenta la diversidad de dominios, formatos y estructuras de los datos.

En esta tesis nos enfocamos en generar algoritmos que aprovechen el conocimiento del dominio para la automatización de partes del proceso de preparación de datos. Mostramos la forma en que las técnicas generales de inducción de programas, en lugar de los lenguajes específicos del dominio, se pueden aplicar de manera flexible a problemas donde el conocimiento es importante, mediante el uso dinámico de conocimiento específico del dominio. De manera más general, sostenemos que una combinación de enfoques de aprendizaje dinámicos y basados en conocimiento puede conducir a buenas soluciones. Proponemos varias estrategias para seleccionar o construir automáticamente el conocimiento previo apropiado en varios escenarios de preparación de datos. La idea principal se basa en elegir las mejores primitivas especializadas de acuerdo con el contexto del problema particular a resolver.

Abordamos dos escenarios. En el primero, manejamos datos personales (nombres, fechas, teléfonos, etc.) que se presentan en formatos de cadena de texto muy diferentes y deben ser transformados a un formato unificado. El problema es cómo construir una transformación compositiva a partir de un gran conjunto de primitivas en el dominio (por ejemplo, manejar meses, años, días de la semana, etc.). Desarrollamos un sistema (BK-ADAPT) que guía la búsqueda a través del conocimiento previo extrayendo varias meta-características de los ejemplos que caracterizan el dominio de la columna. En el segundo escenario, nos enfrentamos a la transformación de matrices de datos en lenguajes de programación genéricos como R, utilizando como ejemplos una matriz de entrada y algunas celdas de la matriz de salida. También desarrollamos un sistema guiado por una búsqueda basada en árboles (AUTOMAT[R]IX) que usa varias restricciones, probabilidades previas para las primitivas y sugerencias textuales, para aprender eficientemente las transformaciones.

Con estos sistemas, mostramos que la combinación de programación inductiva, con la selección dinámica de las primitivas apropiadas a partir del conocimiento previo, es capaz de mejorar los resultados de otras herramientas actuales específicas para la preparación de datos.

Resum

El procés de ciència de dades és essencial per extraure valor de les dades. No obstant això, la part més tediosa del procés, la preparació de les dades, implica una sèrie de transformacions, neteja i identificació de problemes que principalment són tasques manuals. La preparació de dades encara es resisteix a l'automatització en part perquè el problema depén en gran manera de la informació del domini, que es converteix en un coll de botella per als sistemes d'última generació a mesura que augmenta la diversitat de dominis, formats i estructures de les dades.

En aquesta tesi ens enfoquem a generar algorismes que aprofiten el coneixement del domini per a l'automatització de parts del procés de preparació de dades. Mostrem la forma en què les tècniques generals d'inducció de programes, en lloc dels llenguatges específics del domini, es poden aplicar de manera flexible a problemes on el coneixement és important, mitjançant l'ús dinàmic de coneixement específic del domini. De manera més general, sostenim que una combinació d'enfocaments d'aprenentatge dinàmics i basats en coneixement pot conduir a les bones solucions. Proposem diverses estratègies per seleccionar o construir automàticament el coneixement previ apropiat en diversos escenaris de preparació de dades. La idea principal es basa a triar les millors primitives especialitzades d'acord amb el context del problema particular a resoldre.

Abordem dos escenaris. En el primer, manegem dades personals (noms, dates, telèfons, etc.) que es presenten en formats de cadena de text molt diferents i han de ser transformats a un format unificat. El problema és com construir una transformació compositiva a partir d'un gran conjunt de primitives en el domini (per exemple, manejar mesos, anys, dies de la setmana, etc.). Desenvolupem un sistema (BK-ADAPT) que guia la cerca a través del coneixement previ extraient diverses meta-característiques dels exemples que caracteritzen el domini de la columna. En el segon escenari, ens enfrontem a la transformació de matrius de dades en llenguatges de programació genèrics com a R, utilitzant com a exemples una matriu d'entrada i algunes dades de la matriu d'eixida. També desenvolupem un sistema guiat per una cerca basada en arbres (AUTOMAT[R]IX) que usa diverses restriccions, probabilitats prèvies per a les primitives i suggeriments textuals, per aprendre eficientment les transformacions.

Amb aquests sistemes, mostrem que la combinació de programació inductiva amb la selecció dinàmica de les primitives apropiades a partir del coneixement previ, és capaç de millorar els resultats d'altres enfocaments de preparació de dades d'última generació i més específics.

Acknowledgements

Six years ago, I started working as a researcher at Universitat Politècnica de València. To do my research, I had to learn about different fields, not only from computer science but also including environmental chemistry. I enjoyed so much the work that I told my parents: "I'm being paid for being happy". For that reason I started a PhD two years later. However, I soon realised that I was not prepared for what awaited me, neither academically nor mentally. The PhD has been a long and frustrating path of never stopping learning and making lots of mistakes. I thought several times about leaving it, but I never liked giving up. So, here we are: I got it.

Obviously, I would never have imagined ending up this way: writing the thesis completely alone and defending it online because of the lockdown, in the middle of a global pandemic of covid-19. However, that makes this thesis even more special.

Many people have shared moments with me during this time, for better or for worse. Here, I will only name those people who have helped me during this process, either on an academic level or on a personal level. To everyone else, thank you.

First of all, I would like to thank my thesis supervisors for their help over the years. To Cèsar, thank you for letting me know the "research world" and open the doors of the DMIP group for me. The enthusiasm for research that you transmitted to me was unequivocally what convinced me to take this path. To José, thank you for all the support throughout this process; for guiding me when I was lost; for talking when I didn't know what to say; and for explaining everything I didn't understand. Without your help and dedication, I would not have made it. Thank you both for spending time on every little detail of the process (including my never-ending English mistakes) and your infinite patience with me.

Thanks also to Luc de Raedt, for allowing me to do two stays in Belgium within his group. Working in the DTAI group has been enriching both academically and personally. Thank you very much for the opportunity.

To Andrew Cropper, Sebastijan Dumančić and Raul Santos —external evaluators of the thesis—; thank you for your comments and suggestions that have helped to improve this manuscript significantly. To María Alpuente, Fabrizio Riguzzi and Sebastijan (again) —jury of my defence—; thanks for all the questions and comments and to award me with the highest mark.

I would also like to thank all the people in the ELP group, in which I have worked all these years. In particular, I would like to thank María José Ramírez for all her help, both academically and personally, and for collaborating on my papers and giving me the opportunity to teach the data science workshop with

her. María José, you are the most caring and attentive professor I have ever met in my life.

Special mention to all my lab colleagues with whom I have spent most of the time. Julia, thanks for trusting me, making my days happier and always checking that the calendar is not leaning. To Sergi, Ángel and Raül, thanks for all the laughs, bravas and runs. To Nando and David, thanks for the time we have spent together inside and outside the department; for the projects that have been born from the whiteboard; for the trips, dinners, dips with the unicorn, cocktails in Benicassim, cakes for depression and padel afternoons. Thank you for sharing with me (almost) all these years and for making me laugh so much. This path would not have been the same without you. And, finally, to Nacho. Thanks for the gossip afternoons at the lab; for the walks on the beach; the nights of Worms and Diablo III; and for being virtually with me in the lockdown and supporting me when I felt so alone.

I cannot name all the people I've known travelling, but I would like to especially thank the people with whom I have spent most of the time in Belgium. Gust, thank you for collaborating with me and trying to understand me. Ondrej, thanks for all the conversations in the lab, for reminding me the lunchtime and advising me to stop working in the evening, and, of course, for being my personal photographer in Brussels. Evgenya, many thanks for being an unexpected friend when I most needed one. To both of you, thank you for celebrating happiness with me and dancing until dawn. And finally, to Laura. Thank you for the sightseeing; the ice creams on the boat; and for all the conversations we had. You showed up right before it all fell apart and you helped me to stay afloat.

Thanks, above all, to my family and friends who have always been with me. To my brother, Fran; thank you for always being a big support when I need it, a shoulder for crying on and the best playmate. To my Hogwarts' friends: Vero, Arantxa and Ainara, for all the laughs, Asian food, Harry Potter gifts and Scotland memories. To Dani, for being that friend capable of travelling to Belgium when it was more difficult to keep moving forward. Cristina: I don't have enough words to thank you for everything you've done for me since I met you. Infinite thanks for being my friend and my best adventure fellow. Finally, to Jose; for spending these last few months with me in a forced distance, encouraging me to keep moving forward every day. Whatever happens, thank you.

Finally, the most important: my parents. Thanks for supporting me at all times, unconditionally, independently of the distance and always with love. Thank you for always being there, regardless of whether things have gone wrong or right. Thank you for supporting my decisions, even if they ended up being wrong. Thank you for encouraging me to continue or advising me to stop. Thank you for not dropping me and helping me to stand up. I wouldn't have got this far without everything I've learnt from you.

And for me, this deserved smile :)

Lidia Contreras Ochando
Valencia, January 2021

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

"If knowledge can create problems,
it is not through ignorance that we can solve them."
Isaac Asimov

1.1 Motivation

Consider the situation where a data expert should analyse a dataset (see Table 1.1). One of the columns in this dataset includes dates that several users have entered manually, without following any standard (i.e., each of them is presented in a different format). When the expert introduces the dataset into a data analysis tool, it returns an error: "the data type of the column *'Date & Time'* is not recognised". This tool includes an option to convert data belonging to known types into standard formats, so the expert —knowing that all the data on the column are dates— decides to use it to convert all the dates into a unified standard format. The tool transforms a few of them, and returns another error: "some rows are not recognised as standard date formats". It is possible that this problem could be easily solved using some programming language. Nevertheless, the expert does not have programming expertise, so she has no choice but to transform the rest of the dates by hand. This should not be a big problem when the dataset includes only a few rows, but it could end as a really overwhelming task for datasets with thousands of rows and columns. In these cases, the data expert will end up wasting most of her time just transforming all the data into a unified format.

It is widely recognised that a great proportion of the time of data science projects is taken up to carry out data preparation tasks, such as acquiring, understanding, cleaning and transforming the data [30]. Data wrangling is a step inside the data science process that usually involves these tasks. Increasingly often, the available data we are analysing are messy, diverse, unstructured and incomplete, which makes their analysis more difficult. Consider again the data shown in Table 1.1. This table contains some information 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.

Thus, a data wrangling process is usually needed before using the data in most applications of data science, machine learning, databases, etc., where the data are typically provided in a spreadsheet format that has to be formatted. Unfortunately, even when data wrangling is essentially a programming problem [70] (the process could be made easier using data analysis programming languages, such as R or Python), most users lack programming expertise [85]. Programming

requires multiple skills and the learning process of these languages to simply write a small snippet of code can be long and frustrating. Transforming data is usually tricky and the composition of the right primitives (using the appropriate libraries) to get a correct transformation is not always easy.

Can we understand how humans solve this problem so that we can automate the process? Michael Polanyi [143] observed in 1966 that humans perform many tasks without knowing very well how we do them. Trying to explain how to automate data wrangling tasks is an almost impossible work. Nevertheless, recent researches have shown that deep learning is able to automate many tasks from other areas [35, 102, 153] by using a huge quantity of data. For example, the problem of face recognition: any human is able to recognise the faces of friends in a crowd of people, but this is not an easy task to explain how to do it. Deep learning is able to recognise faces based on the characteristics learned by studying thousands of faces (shapes, colour, nose, eyes, etc.). According to Polanyi [143], the type of tasks that we understand only in a tacit way has proved to be the most difficult to automate, specifically those demanding judgement, common sense, creativity and experience. Recent studies using different methods have similar conclusions analysing that some jobs are more susceptible to computerisation than others [3, 39, 40, 52, 86]. Some of those jobs difficult to automate are, for instance: "collect, organise, interpret, and summarise data to provide usable information"; "compile and compute data according to statistical formulas to use in statistical studies"; or "create, modify, and test the code, forms, and scripts that allow computer applications to run". This means that, inside the data science process, one of the most difficult steps to automate is data wrangling.

Despite the increasing efforts for automating some other processes included in data science, such as machine learning [67], the manual tasks of data wrangling still resist fully automation, partly because the problem strongly depends on domain information. If we really want to automate data science we need to know how data scientists behave and learn from them. For instance, if two different dates are to be formatted into a different format to be properly recognised by a data analytics tool, humans usually process this in two steps: (1) they recognise that the problem is related to dates and (2) they apply specific conversions for dates. However, these conversions are specialised for application domains. For instance, the mechanisms to manipulate dates are very different from those

Name	Address	Date & Time	Country
Alejandro Pala C.	C/Jose Todos, 22	03/04/17 19:39	Spain
Clau Bopper	Rua bolog, 136	27/06/2017 22h56	France
Srta. Maria Garcia	Av. Del Mar 14, piso 6, 12	4 octubre 2017 10:20	Mexico
Dr Lauren Smith	Flat 5, Royal Court, Coventy	30 October 2017 9:45	UK
Sabrina Bunha P.	Rua Beni, 365, Alegre	27/11/2017 07h05	Brasil
Mr David Bozz	88 Lane of trees, Texas 77925	12/21/2017 12:30 PM	USA

Table 1.1. Example of personal data presented in different formats depending on the user's geographical region.

to manipulate addresses or any other type of data. Consider Table 1.2. The difficulty of this problem lies in the different date formats, where the day can be the first, second or third number. Additionally, these numbers can be delimited by different symbols.

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

Table 1.2. 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.

A second problem that slows down the progress in the automation of these data science processes is the fact that there is rarely a public repository of raw, messy or unprepared data to work with, since most of the publicly available datasets have been already transformed or pre-processed. Besides, the work of cleaning and preparing the data is not usually fully detailed or documented [133]. It becomes very difficult to automate a task when it is not well described or defined, and also lacks enough examples for learning how to do it.

Recently, Inductive Programming (IP) [49, 62] has been successfully applied to the automation of data transformation problems [5, 59, 101, 144]. Inductive reasoning makes generalisations from specific observations, i.e., it takes observations (a few examples), discerns a pattern, makes a generalisation, and infers an explanation or a theory (rules on some programming language, generally declarative). To do this, inductive programming uses a declarative[1] background knowledge (BK).

It should be noted that all of these recent IP systems are based on domain-specific languages (DSLs), i.e., languages that are defined for a particular kind of processing (e.g., string processing, number processing, etc.). If we choose some of these domain-specific systems, e.g., one specialised for string processing, to solve the problems on Table 1.2, they may never find the right solution. The trouble for these systems is that their basic functions do not allow them to really know what the real problem is: Extracting the first number? Two digits? Or everything before any symbol? In order to solve this problem we must know how the domain works, its constraints and how it is usually represented. This means that solving the diversity of problems from each different domain will require the appropriate background knowledge. As a result, too much information can become a bottleneck as the diversity of domains and formats increases, since

[1] The functions or primitives in the background knowledge are expressed using declarative languages.

when the set of primitives becomes too large the search for a suitable combination will become almost intractable [42, 73].

One alternative to DSLs is general-purpose declarative (programming) languages (GPDL). These languages are the base for most of the inductive programming systems created during the last two decades, such as Progol [128], MagicHaskeller [90], FLIP [45, 72], Metagol [131], gErl [116] and many others. Unfortunately, inductive programming using GPDLs is usually inefficient and/or incomplete because of the large search space, as they are not specialised for a particular domain. Nevertheless, with the correct definition of a reduced library of functions (or predicates) in a domain-specific background knowledge (DSBK), the search space for these generic inductive programming tools can be bounded by the size of the solution and the number of functions in this DSBK. In other words, in theory, the use of inductive programming using GPDL + DSBK can be as powerful as the use of inductive programming tools that are specifically designed for a particular DSL. The advantages of using this approach are manifold. First, the same data wrangling tool with inductive programming can be used for a diversity of problems and domains, without specialised tools for every domain. Second, a library of DSBKs could be provided with the data wrangling tools using inductive programming. The user just needs to suggest which one to use for a particular problem: dates, times, emails, names, cities, addresses, etc. Since the languages for creating the DSBK are general-purpose and well known (Haskell, Prolog, etc.), users can create their own DSBKs and share them with other users to help them automate their data wrangling transformations.

Every problem related to data preparation can become easy to automate by using the appropriate background knowledge, but the number of domains can make the size of the space of solutions too large, making the problem difficult to be solved with current systems. The answer to this issue can lie in the use of general purpose languages together with IP systems but dynamically finding the correct piece of the background knowledge to adapt the solution to each particular problem. Finally, if we really want to automate or semi-automate this process, we should require the less intervention of the user as possible, i.e., we need to solve the problem by using only a few examples from the user.

1.2 Research Questions and Objectives

Automating data wrangling would be tremendously useful. However, as we have seen on the introduction, there are some issues that do not allow the process to be completely automatic, such as a lack of documented examples of data wrangling problems. Inductive programming is capable of learning, even when there are few examples, but a sufficient domain knowledge is needed to be able to solve different problems. The problem for this paradigm is that, if the background knowledge is too large, it becomes a bottleneck that slows down —or even makes intractable— the inference of the solution.

With these problems in mind, this thesis will try to answer the following questions:

- What are the challenges for an algorithm to correctly and efficiently automate the data transformation process independently of the data domain?
- How can we dynamically select the background knowledge of an inductive programming system to find the solution to the problem?

The aim of this thesis research is to **provide algorithms and inductive programming systems capable of dynamically select the appropriate background knowledge —depending on the problem to solve— when dealing with data wrangling tasks, but requiring the lowest user intervention possible (i.e., automatically or semi-automatically).**

To achieve this main goal, the following objectives will be accomplished:

- Analyse the role of data wrangling and its automation in the trajectories of data science projects.
- Study which data domains are normally used in data cleansing tools and what characteristics they have.
- Acquire sufficient examples of common problems related to the transformation of data belonging to the studied domains in different scenarios.
- Generate a domain-specific background knowledge for each domain, large enough to be able to solve as many problems as possible.
- Generate algorithms to dynamically handle the background knowledge, regardless of its size, for a range of data wrangling problems and domains, requiring the least possible time.
- Provide systems based on inductive programming, which use the algorithms and the generated background knowledge, in order to solve the collected problems, using as input as few examples as possible.

1.3 Research Methodology

To reach the goal described in the previous section, the Design Science Research (DSR) methodology [78] has been applied, which is fundamentally a problem-solving paradigm. The goal of this methodology is to produce a viable artefact such as algorithms, human/computer interfaces, design methodologies (including process models) and languages, and develop knowledge that other professionals and researchers of the field can use to design new solutions. In this respect, all the experiments, code and data have been published in public repositories shared with the community to allow for further experiments and repeatability.

Following the DSR methodology guidelines [167] this thesis has followed the following steps:

1. **Identification of the problem:** first, a research on the state-of-the art systems for data wrangling has been carried out. The goal of this step is to find the tools developed today that help on transforming data in a manual, semi-automatic or automatic way and which are the advantages of using

them as well as their limitations. With this information it is possible to define what kinds of problems prevent us from automating data wrangling process.

2. **Definition of objectives:** if we want to automate data wrangling tasks the main problem to solve is to deal with all the different domains and data formats. After the study on the current systems and as stated in section 1.2, this thesis attempts to explore new ways of reducing the size of the background knowledge for the inductive inference when dealing with data wrangling automation.

3. **Research design:** when the objective is clear, the research has to be designed. In this case, we have done this in three steps: (1) we have made a study to know the different existing inductive programming systems in order to choose which is the best one to use for our purpose, as well as the language in which they are implemented; (2) after the study of the state-of-the art tools for data wrangling, we have compiled a list of common problems related to data transformation that are repeated in different systems and asked in several help forums. These kinds of problems (for instance, transforming a date into a specific format) are real problems when working with data that can be simplified by automating the process; (3), with the list of problems we have put together a collection of data from several sources in order to have real problems to work with. The following data sources have been used:

   - **Personal data collected through surveys:** we have created a survey[2] in order to collect personal data (name, surnames, address, phone, etc.) from people. The goal of this survey is to collect data in very different formats depending on the user's geographical location. The survey has been open on Twitter for a month and in total we have collected 59 responses. After the collection of these data an anonimisation process has been performed to the data.

   - **Data collected from the literature:** as we will explain in section 5.4, we have also collected most of the datasets tested previously in other tools for data manipulation in the literature [6, 41, 59, 156, 157].

   - **Data generated by pattern:** Following the patterns from the data collected in the above points but changing the values, we also have generated new examples.

   - **Data collected from help forums:** we have collected examples related to data transformation from help forums such as StackOverflow[3], since they are a really good place to look for real problems.

[2]The survey can be found in Appendix A.
[3]StackOverflow: https://stackoverflow.com/

- **Synthetic data:** finally, we have also generated some examples of random matrices for the experiments[4] of Chapter 6.

And finally, (4), we have developed the theoretic formulation of the solutions, we have designed the algorithms (by defining the inputs, outputs, instructions, constraints, etc.) and we have performed a preliminary analysis (both theoretical and experimental).

4. **Development of artefacts:** with all the information collected and the studies done, we have finally developed two different systems to automate different data wrangling problems: (1) The first one has been implemented using an existing IP system as the back-end that allows us (and future users) to generate new transformations of data in a general-purpose declarative (programming) language; (2) The second system (created from scratch) implements an algorithm in a common statistical language, which allows us to generate a package that future users can download easily.

5. **Evaluation of the solution:** the systems have been tested with the collected examples (using only one —full or partial— example as input) and evaluated following different criteria: (1) we have measured the accuracy of the systems using the number of correct solutions obtained; (2) we have measured the time spent to solve the problems; (3) we have measured the size of the background knowledge or the number of solutions explored for each problem solved with the different strategies presented; and (4) we have compared the results obtained with the results obtained with other existing tools.

6. **Communication of the problem and the solutions:** we have shared the results and data used in three different ways: (1) We have developed a new data wrangling dataset repository including the data collected or generated; (2) we have created a GitHub repository with the code of the systems; and (4), all the methods and results shown in this thesis have been published and presented in different venues (conferences, journals, seminaries, summer schools and other meetings) in different formats (papers, posters, presentations and demos).

1.4 Thesis Outline

The dissertation has been divided into five parts, being the first one the preliminaries that include this introductory chapter. The second part includes two chapters describing the fundamentals and covering the state-of-the art through the literature of the main fields used for the development of this thesis: inductive programming and data science. The two main research contributions

[4]The criteria followed to gathered the data of this bullet point and the bullet point above is presented in section 6.4.

of this work are described in parts three and four. Finally, part five closes the thesis with the conclusions and future work.

A detailed description of each chapter is provided below:

- Part II: Background

  - Chapter 2 summarises the basic concepts of artificial intelligence, machine learning and inductive programming. The chapter also presents the process of inductive inference and discusses the problem of the inductive bias when too many primitives are added to the hypothesis space in program synthesis. Finally, the chapter describes some general-purpose inductive programming systems and the functionality and theory behind the inductive functional programming system, *MagicHaskeller*, used as the core of the BK-ADAPT system. The chapter also analyses how the inductive bias can affect *MagicHaskeller*.

  - In Chapter 3 the process of data science is presented and the progress and challenges in the automation of some of ts activities, data wrangling in particular. We follow the examples of [114] to describe data science as a space of activities where many of them can occur, once or repeated times in the same project without following a determined order or path. Then, the chapter focuses on one of the data science steps: data wrangling. Data wrangling automation, which is the main goal of this thesis, is described in different sections. First, the issues of its automation are exposed as well as some possible solutions. Then, the full data wrangling process and its issues are presented following the criteria of [133]. The chapter analyses the current research lines for automating data transformation and the future challenges of this field. This chapter is partially based on the following publications:

    * **Logging Data Scientists: Collecting Evidence for Data Science Automation.**
      Lidia Contreras-Ochando, Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, María José Ramírez-Quintana.
      AI4DataSci @ NIPS, 2016.

    * **Applying the Skills in Data Science with Those in AI/ML.**
      José Hernández-Orallo, Lidia Contreras-Ochando.
      Dagstuhl Seminar 18401 Automating Data Science, 2018.

    * **CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories.**
      Fernando Martínez-Plumed, Lidia Contreras-Ochando, Cèsar Ferri, José Hernández-Orallo, Meelis Kull, Nicolas Lachiche,

María José Ramírez-Quintana, Peter Flach.
IEEE Transactions on Knowledge and Data Engineering, 2019.

- Part III: BK-ADAPT: Automating Data Feature Transformation

  – In Chapter 4 the domain-specific background knowledge approach is included. This is used to solve the problem of having a big set of functions when several domains have to be added to a data transformation system. Here, the data wrangling dataset repository is created with six different domains and used to evaluate whether the use of a domain-specific induction can be useful to improve the search of the right primitives. This chapter is based on the following publications:

    * **General-Purpose Inductive Programming for Data Wrangling Automation.**
    Lidia Contreras-Ochando, Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, María José Ramírez-Quintana.
    AI4DataSci @ NIPS, 2016.

    * **Domain specific induction for data wrangling automation.**
    Lidia Contreras-Ochando, Cesar Ferri, José Hernández-Orallo, Fernando Martınez-Plumed, Marıa José Ramırez-Quintana, Susumu Katayama.
    AutoML @ ICML, 2017.

    * **Domain specific induction for data wrangling automation.**
    Lidia Contreras-Ochando.
    Dagstuhl Seminar 17382 Approaches and Applications of Inductive Programming, 2017.

    * **General-purpose Declarative Inductive Programming with Domain-Specific Background Knowledge for Data Wrangling Automation.**
    Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, Susumu Katayama.
    arXiv preprint arXiv:1809.10054, 2018.

  – Chapter 5 introduces the BK-ADAPT system. BK-ADAPT is a system able to transform data that are presented in very different formats for a domain. Here, the dynamic background knowledge approach is presented. This system uses the domain-specific induction together with a machine learning meta-model that is feed with meta-features extracted from the data to estimate the functions needed to

solve each specific problem. This chapter is based on the following publications:

  * **Automated Data Transformation with Inductive Programming and Dynamic Background Knowledge.**
    Lidia Contreras-Ochando.
    Dagstuhl Seminar 19202 Approaches and Applications of Inductive Programming, 2019.

  * **Automated Data Transformation with Inductive Programming and Dynamic Background Knowledge.**
    Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, Susumu Katayama..
    European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD), 2019.

  * **BK-ADAPT: Dynamic Background Knowledge for Automating Data Transformation.**
    Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, Susumu Katayama.
    European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD), 2019.

- Part IV: AUTOMAT[R]IX: Automating Matrix Transformations

  – In Chapter 6 the AUTOMAT[R]IX system is presented. First, the chapter describes the problem that is represented by programming in the context of data transformation for people with a lack on programming skills. As a case of study the matrix transformation problem is illustrated. Then, the chapter describes the method and the algorithm created for AUTOMAT[R]IX using a probabilistic approach that uses the dimensions of the input/output matrices, prior information in the form of frequency of use of the functions and some (optionally) tips from the user in the form of natural language. Finally, the chapter shows the experiments performed and results obtained. This chapter is based on the following publications:

    * **Automating Common Data Science Matrix Transformations.**
      Lidia Contreras-Ochando, Cèsar Ferri, José Hernández-Orallo.
      ADS @ ECMLPKDD, 2019.

    * **AUTOMAT[R]IX: Learning Simple Matrix Pipelines.**

Lidia Contreras-Ochando, Cèsar Ferri, José Hernández-Orallo. *Machine Learning, 2020 (to appear).*

- Part V: Conclusions
  - Finally, Chapter 7 closes the thesis with the conclusions, main contributions and possible directions for further research in the field of data wrangling automation.

At the end we have also added some appendices with further information:

- Appendix A includes the survey used to collect personal data.
- In Appendix B we describe the competence groups that identify the main competences and skills required in data wrangling process.
- Appendix C describes some ideas for logging data scientists in order to learn from their behaviour.
- Appendix D includes some supplementary material for the BK-ADAPT approach, including the extended results and the description of the system architecture.
- In Appendix E we also include supplementary material for the AUTOMAT[R]IX approach and a pipeline of the events with a simple example.

Part II

Background

Chapter 2

Inductive Programming

In this chapter, we describe the fundamentals and goals of inductive programming. We discuss how to deal with a large space of solutions that is dominated by the inductive bias. The chapter also includes a description of *MagicHaskeller*, a general-purpose learning system used as the core of the BK-ADAPT system presented in this thesis.

This chapter is organised as follows. Section 2.1 summarises what artificial intelligence, machine learning and programming induction are and explores how machines learn. This section also explains the random forest technique that will be used in the following chapters. The basis of the inductive inference of programs is illustrated in section 2.2 and the the problem of inductive bias is explained in section 2.3 in the context of programming by example, a kind of inductive programming. Finally, section 2.4 lists some of the inductive programming systems and ends with section 2.4.1 presenting the inductive functional programming system *MagicHaskeller*.

2.1 Introduction

In 1950, Alan Turing proposed that machines can 'learn' [165]. Following Ada Lovelace's objection that machines can only do what they have been told to do [118], he stated that machines could learn if they were programmed to imitate a child's mind. This way, we could teach machines with experiments that exercise their "minds" by an education process. Turing established the fundamental goal and vision of Artificial Intelligence (AI). AI is the science of making machines perform human tasks on their own, by the simulation of human intelligence and human abilities. The goals of artificial intelligence include learning, reasoning, and perception [51]. A few examples of applications of AI are: Google assistant[1], Siri[2], Alexa[3] and other personal assistants; self-driving cars; or IBM's Watson[4].

In order to make AIs able to "think" on their own, we need some techniques to teach them how to learn. Machine Learning (ML) is a specific subset of AI that trains a machine on how to learn using data. In machine learning, machines are not specifically programmed to do a task, instead, some data is given to them to learn how to do it. Then, the machines look for patterns in the data and try to draw conclusions. The result of this is a machine learning model. As Peter Flach defines in [46]: "Machine learning is the systematic study of algorithms and systems that improve their knowledge or performance with experience".

[1] Google assistant: https://assistant.google.com/
[2] Siri: https://www.apple.com/siri/
[3] Alexa: https://developer.amazon.com/es-ES/alexa
[4] IBM's Watson: https://www.ibm.com/watson

Machine learning is closely related to the fields of statistics and data mining [66, 132] and is used in a wide variety of applications, such as email filtering and computer vision, where it is difficult to develop conventional algorithms to perform the needed tasks.

There are many different ways in which AI systems can learn, and many machine learning techniques have tried to cover a wide range of possibilities. One important dimension to characterise all these techniques would be how data-intensive they are. Here, the spectrum goes from models that are constructed in a data-driven way (i.e., making decisions based on data analysis and interpretation), to models that are built in a theory-driven or compositional way (i.e., the hypotheses are made from theories that explain a behaviour) [54, 100]. The former are now predominant in machine learning because they are appropriate for large volumes of data and work well when we do not know the appropriate bias information for each domain (or we are unable to put that bias into the systems). The latter are usually better for learning from a few examples, especially if the information about the domain can be incorporated as background knowledge composed of a set of auxiliary primitives (functions) or concepts as explained in section 1.1. Inductive Programming (IP) [49, 62] is a clear example of this family of techniques. Inductive programming learns theories (logic, functional, or functional-logic programs) from incomplete specifications, such as a few input/output examples, possibly using a declarative background knowledge that works as a powerful explicit bias to reduce the search space and to find the right level of generalisation. However, there are scalability issues since the hypothesis space grows exponentially with the size of the background knowledge. However, we find that most IP systems are only able to select the functions of the background knowledge by simply considering the appropriate types of the arguments, in the best of cases, but not according to the problem at hand.

2.1.1 How Machines Learn

In cognitive science, learning is the process of gaining knowledge through observation. We need to have some prior knowledge related to a task in order to do it in a proper way. For instance, when children learn how to add they can easily transfer this knowledge to better understand and learn how to multiply. Besides, if we keep learning and gaining more knowledge about the task we can improve and perform it more efficiently [154]. Human learning can happen in three ways: (1) we try to do it ourselves until we success (search, self-experience), (2) someone expert in the task teaches us and guides us (demonstration), or (3) we learn by knowledge generated from others in the past (transmission) [74].

So, can these types of learning be applied to machines? Tom M. Mitchell defines that "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [121]. That is, a machine can learn if it is able to collect experience (from training data) by doing a task and improve its performance doing it. It can also do other similar tasks in the future (generalisation). The primary elements of machine learning are features

(the description of the objects), tasks (the problem we want to solve regarding the features) and models (output of a machine learning algorithm applied to data) [46].

Overall, there are three main types of learning related to machine learning (though hybrids do also exist), depending on the nature of the feedback available to the learning system [103]:

- **Supervised learning:** The computer is given the example inputs and their expected outputs. The goal is to learn a general rule that maps inputs to outputs. Basically, we are telling the machine "Do like I do". We call these tasks *classification* when the labels (outputs) are categorical values, and *regression* when they are quantitative values.
- **Unsupervised learning:** No labels are given to the algorithm, so it has to find the relations in its input. Unsupervised learning can be used for discovering hidden patterns in data or for feature learning. In this case, we say to the machine "Find patterns in the data".
- **Reinforcement learning:** The algorithm interacts with a dynamic environment used for learning how to behave when given occasional reward or punishment signals while the algorithm must perform a certain goal. So, here we are telling the machine "Do whatever you want to get the highest score".

Other types of machine learning are: active learning, which lets users play an active role in the learning process by asking them to label examples; and semi-supervised learning, where the data are a mixture of labelled and unlabelled examples [66].

2.1.1.1 Feature-based learning techniques: random forest

Many supervised machine learning techniques map inputs to outputs in nonlinear ways where the goal is to approximate the mapping function so you can predict the output variables from new input data. The learning stops when the algorithm achieves an acceptable level of performance. For the purpose of our work we will use a supervised machine learning technique (Random forest). In this section we will describe this technique with more detail. Supervised models estimate unknown future values. Inside supervised models we can find classification and regression tasks [44, 137]. Figure 2.1 summarises how supervised machine learning methods work. In the supervised approach, the objective is to learn a mapping from inputs x to outputs y, given a training set $D = \{(x_i, y_i)\}_{i=1}^{N}$, where N is the number of examples. Each training input x_i is a feature, attribute, covariate of features or a complex structured object (such as an image, a sentence, etc.). The supervision comes from the labelled examples in the training data set. [132].

In order to understand how the random forest technique works, we first need to introduce how decision trees are learned. Decision tree learning is one of

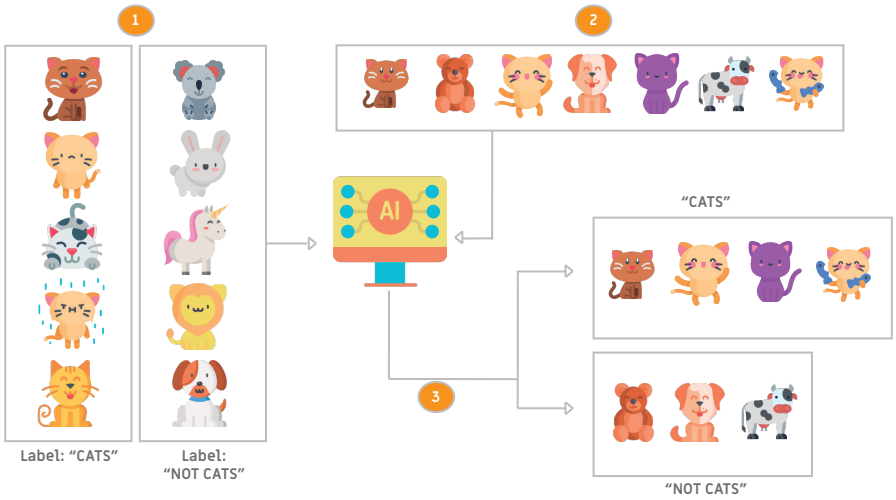


Figure 2.1. Supervised machine learning: (1) The machine learning algorithm is given labelled data for learning; (2) The algorithm is provided with new unlabelled data; (3) The algorithm tags the new data according to the information learnt in the first step. Adapted from [15].

the oldest and most popular techniques in machine learning and statistics [11, 82, 89, 147]. These methods construct a model based on conditions on the attribute values organised as a tree. Each node of the tree corresponds to one of the features and from every node, there are edges (branches) for each of the possible values of the feature. Each leaf node represents a possible value for the output variable (label). The decision tree can be turned into a logical expression. Decision tree learning is not limited to classification; it can be used in almost any machine learning task (such as regression and clustering).

Random forest is an ensemble model, i.e., a combination of models. Ensemble models are one of the most powerful techniques in machine learning. One way of building an ensemble model is by training models on random subsets of the data. These random samples taken from the data are known as bootstrap samples (bagging). The number of features for each sample is a 'hyperparameter' that is set to some percentage of the total. For a sample of size n, the probability that a data point is not selected is $(1 - 1/n)^n$ (i.e., each bootstrap sample leaves out approximately a third of the data). Random forest uses decision trees as the base classifier and their models that are independently trained [46]. Each tree includes a random sample from the original data set, adding randomness that prevents overfitting.

The predictions of the models returned are combined (they can be combined by many methods, including averaging, voting, and probabilistic methods [46]), treating them as a "committee" of decision makers to make the overall prediction. This committee of decision should have better overall accuracy, on average, than

any individual committee member (see Figure 2.2).

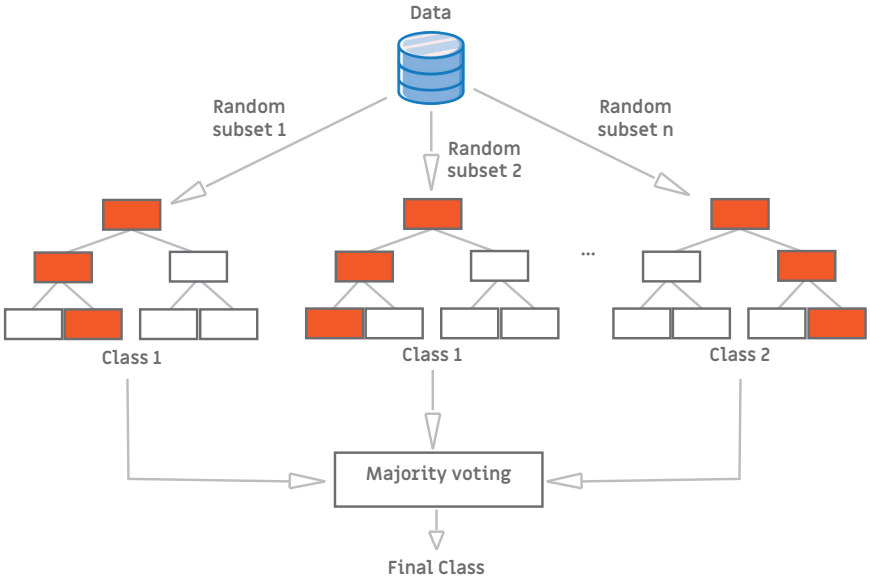


Figure 2.2. Ensemble Algorithms: Random Forest. The class is selected with the majority of the votes from all the trees. Adapted from [12].

2.2 Learning programs

Many machine learning models are represented as mathematical entities, linear or non-linear models with weights that connect inputs and outputs. Some of them are expressed in terms of distances, or even represented as rules. But we can also think of a model as a program. When taking this perspective, the goal is to build a program, but not by writing the instructions from a specification. Instead, the program is built by inductive inference, following a few examples of what the program has to do. Inductive Programming —also known as program induction, example-based programming or inductive program synthesis—, is a sub-field of machine learning and an inter-disciplinary domain of research, of which goal is to construct a computer program or algorithm from an incomplete specification (only some parts are specified) of a target function. As Figure 2.3 shows, in contrast to deductive inference —where a set of rules (theory) have to be applied to predict the observations [47, 109, 110]—, inductive inference makes generalisations (discoveries) from specific and incomplete observations (called the evidence) by identifying general patterns in the data. Such observations can be positive and negative examples, clausal constraints, uses cases, desirable behaviour of a software, computational traces, etc. [8, 50, 96].

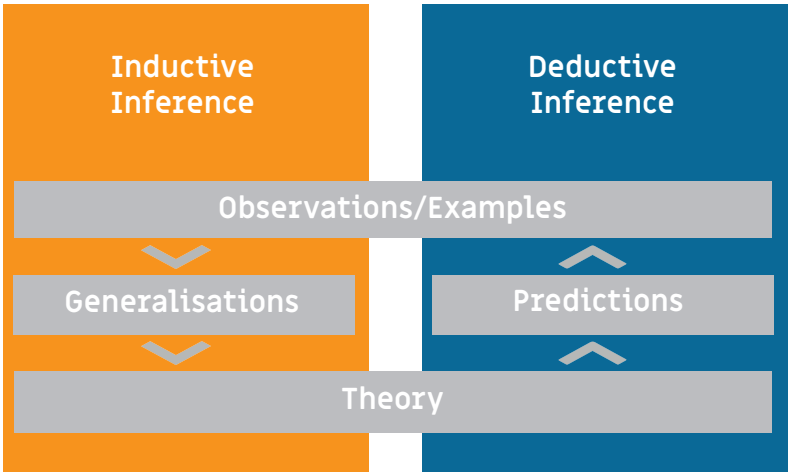


Figure 2.3. Inductive reasoning vs deductive reasoning.

The inference can be based on background knowledge or the learner can query an oracle (the user) [49]. Depending on the programming language used, there are three types of inductive programming: (1) Inductive functional programming [7], (2) inductive logic programming [127] and (3) inductive functional logic programming [77]. An induced program contains function primitives, i.e., predefined functions known by the IP system or given as a background knowledge. However, the hypotheses are not always easy to construct. On the one hand, the hypothesis space can be huge and complex when the background knowledge has many primitives, resulting in scalability issues because of a combinatorial explosion. On the other hand, there is a problem if the used language is not expressive enough to represent the target hypothesis.

For humans it is easier to give examples of a target concept than to program it [164]. Inductive programming can be applied to many areas such as evolutionary programming, grammar inference, programming languages and verification [75]. Furthermore, inductive programming approaches are of interest in programming by demonstration applications for end-user programming as well as in cognitive models of inductive learning. Recently, it has been shown that the capability of inducing programs from examples makes inductive programming also suitable for many real-world applications [62]. Programming by example (PbE) [104, 149] is a sub-field of inductive programming, where the specification comes in the form of input-output examples [60]. The user provides examples of data before and after processing; the computer must find a function for transforming the input to the output. Generally, in order to make PbE widely useful, only a few examples should be sufficient to induce the right solution even when the examples are incomplete. The use of declarative languages allows the synthesis of programs including recursion or loops, as well as primitives from a declarative

background knowledge [50, 76, 97, 129]. In particular, PbE can help to automate many data manipulation tasks, such as extracting terms from texts, transforming data format, inverting strings, etc. Since 99% of people who use computers lack programming skills [81], they often spend considerable time and effort performing tedious and repetitive tasks, for instance capitalising a column of names manually. Inductive Programming can liberate users from many tasks of this kind, because it addresses the human act of computer programming by generating programs or scripts automatically [49].

2.3 Inductive Bias

From a machine learning point of view, the selection of functions from the background knowledge can be seen as a kind of bias [121]. Infinitely many (semantically) different programs can meet the specification. Hence, one needs criteria to choose between them. As said before, the background knowledge is usually composed of a set of auxiliary primitives or concepts that can be combined to find the hypothesis that covers the data. However, if this set of primitives becomes too large then the search for a suitable combination becomes huge. As usual, bias makes the learning of some hypotheses easier (or possible) at the cost of other hypotheses. A similar problem appears in other incremental settings when a system learns concepts, representations or features gradually but at some point there are too many to combine [96].

In particular, the term 'inductive bias' refers to the assumptions a learning system does to prioritise some hypotheses over others [122]. In approaches where the hypothesis combines primitives or concepts, the inductive bias has the aim of adapting the depth (d) —how many primitives or elements are needed– and breadth (b) –how many choices there are in the library of components— of the learning process. Thus, with no alteration of the search procedure, the background knowledge can be used to produce a bias on learning. However, as the background knowledge grows to reduce d for more and more problems, the search becomes intractable because of the growth of b. To avoid these problems, inductive programming can use different mechanisms to constrain the search for hypotheses: language bias (determining the hypothesis space, i.e., which programs can be induced) and search bias (determining how to search in the hypothesis space, i.e., determining which generalised programs are constructed first) [9]. If we only provided a few general primitives, d would increase considerably, as these primitives should be combined in elaborate ways to make even simple transformations. However, as more kinds of functions 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, $O(b^d)$ [45, 71], with n being mostly irrelevant. Still, the great advantage of inductive programming is that it can infer a solution for one or a few examples (the benefit of automation disappears if the user has to provide many examples).

Inductive bias has been analysed in incremental and lifelong learning scenarios [45, 120, 123]. The general idea is to combine the hypothesis generation process with a forgetting mechanism to limit the amount of background knowledge that must take part in learning. [115] explains an approach that rate rules from an inductive engine. The algorithm uses a coverage graph to introduce several metrics that allow forgetting some of the worst rules and the promotion of the selected ones. In [28], Cropper uses two different methods (syntactical and statistical) to forget redundant knowledge reducing the background knowledge. The experiments show that forgetting can improve the learning performance and avoid overwhelming by too much background knowledge. In a recent work [38], Dumančić and Cropper claim that re-structuring the knowledge is essential, instead of removing or updating it. The paper argues that re-structuring knowledge can provide a better inductive bias to a learner. In this thesis, we will see a method for restructuring the knowledge in a dynamic way based on domain-specific induction (chapter 4) and on the use of probabilities with a tree based search, that also combine the deletion (or forgetting) of those hypotheses that the system detects that will never become a solution to the problem (chapter 6).

Another idea is to predict which functions may be needed and select only those ones. In [4], Balog et al. train a neural network with a set of generated problems that is able to predict which functions may be needed for a particular problem (presence or absence of individual functions). The results show that the system is able to speed up the process compared with a baseline using prior as function probabilities, although they only include 34 functions in the DSL[5]. To avoid the combinatorial explosion with huge DSLs, the EC^2 algorithm [42] (based on the *E.C.* algorithm, introduced by [33][6]), learns a reduced set of functions through three steps: (1) exploring the space of programs guided by a neural network searching for programs that solve the task; (2) compressing the DSL with domain-specific subroutines; and (3) compiling the DSL using a neural network to write programs. The learned DSL uses the domain as a prior and the input-output example for a specific task as a posterior. Using this approach in the text editing domain, they are able to solve 74% of the problems. In [119], the idea of ranking the functions according to some text features is presented. However, in this work, Menon et al. rely on the fact that input and output strings are closely related. For instance, the output is a sub-string of the input. In chapter 5 we will see an approach to rank the functions included in the background knowledge —based on meta-features extracted from the examples—, that is able to obtain the solution even when the background knowledge includes more than 200 functions.

In order to guide the search thought the hypothesis space, in [135], Oliphant and Shavlik use two Bayesian networks, one trained in "good" predicates and the other trained in all the predicates, to generate a weight that can be attached to

[5]The DSLs are explained in section 1.1.

[6]The E.C. algorithm is able to solve multi-task program induction by learning from simple tasks a search space structure that enables it to solve more complex tasks.

all the candidate predicates. The algorithm runs a hill-climbing algorithm[7] using the weights to guide the search. In this way they are able to use probabilities to select the predicates to apply for each example. Finally, the results shown in [158] suggest the usefulness of a measure of relevance on the background knowledge to guide the search over programs relying on expert knowledge. In this thesis, we will see how the search can be guided by the syntactic structure of the examples (part III) and with the use of tips or descriptions in natural language about the problem that can be provided by the user (part IV).

2.4 General Purpose Inductive Programming Systems

The first attempt to make an Inductive Programming System (IPS) was made by Summers with the THESYS system [160]. He noticed that with certain allowed primitives, a program schema, and some small sets of positive input/output examples, a recursive LISP program can be found after a search in program space. The system synthesised linear recursive LISP programs by rewriting the examples into traces. This led to further research on the topic and several IP systems have been created. For the purpose of our work, we looked for a general-purpose IP system using a well-known declarative language (not DSLs) so we can modify both the system and the background knowledge with the domain knowledge (as we will see in chapters 4 and 5). In this section we summarise some of these systems[8] and the reason why we choose one of them: *MagicHaskeller*. We divide these systems into four groups, as follows:

- **Analytical approaches:** These systems follow a example-driven way, that is, the structure of the examples is used to guide the construction of the programs.

  - *IGOR I* [97]: modern extension of Summer's THESYS system. Uses LISP and the functions can only be first-order (i.e., not taking functions as arguments nor returning functions as result). No nested or mutual recursion is allowed.

  - *IGOR II* [95]: this system relies on constructor-term rewriting techniques. The functions are defined with algebraic data-types and the search is guided by pattern matching. The specifications are presented as sets of example equations that can contain variables and the background knowledge can be provided in form of additional example equations.

- **Search-based approaches:** These systems first construct one or more

[7] Hill-climbing algorithm tries to find a better solution by incrementally varying a single element. If the change produces a better solution, another incremental change is made to the new solution, repeating this process until no improvements can be found.

[8] This list is based on the list of systems published at https://inductive-programming.org.

hypothetical programs, evaluate them with the input/output examples and then work with the most promising hypotheses.

  - *ADATE* [136]: synthesises function definitions in a subset of the programming language ADATE ML, guided by an evaluation function that tests a given program and says how good it is. Inspired by basic biological principles of evolution (genetic algorithms). It is especially suitable for reinforcement learning and it is intended as a system that automatically improves a part of an existing program, covering the examples via a generate and test.

  - *MagicHaskeller* [92]: The system generates HASKELL programs by using type-constraints. It searches the space of λ-expressions (i.e., a function definition that is not bound to an identifier) for the smallest program satisfying the specification. The expressions are generated using function composition with user provided functions and data-type constructors. It uses a breadth-first search over the candidate programs guided by the type of the target function.

- **ILP Systems:** These systems have a focus on learning recursive logic programs.

  - *GOLEM* [130]: This system constructs a set of definite clauses from groups of positive and negative atomic examples together with extensional background knowledge. It starts by randomly choosing pairs of positive examples that are removed once a single clause has been asserted, continuing on all remaining examples. However, such a search space makes search nearly intractable.

  - *PROGOL* [128]: The system uses PROLOG and combines "Inverse Entailment" (i.e., using a background knowledge B, a hypothesis H covers an example e, if and only if $B \wedge H \models e$) with a "general-to-specific search" (i.e., h_1 is more general than $h2$ if and only if $(\forall x \in B)[h_2(x) = 1) \rightarrow (h_1(x) = 1)]$, being h_1 and h_2 boolean-valued functions defined in B) through a refinement graph. It performs an A*-like search, guided by compression.

  - *DIALOGS-II* [48]: This system uses PROLOG and no evidence needs to be prepared in advance. It queries an oracle and invents its own evidence about it. Type declarations are provided as language bias.

- **IFLP systems:** These systems have a focus on learning recursive logic programs

  - *FLIP* [77]: Induction of Functional Logic Programs from facts, based on the reversal of narrowing. It is searched heuristically using a combination of description length and coverage of positive examples.

> Its applications are mainly program synthesis, program debugging and data mining of small highly structured documents.

In [80], Hoffmann et al. compare all these systems with a set of experiments and conclude that there are improvements still to be discovered. However, they show that ILP systems need a higher number of examples than the other systems, while IFP systems get along with much fewer examples and are much more reliable in their results. Finally, they also state that analytic approaches, such as *MagicHaskeller*, have problems when the search space is wide but the ability of generically inventing functions is a big advantage for them.

As we will see in Part III, for the BK-ADAPT system we decided to use *MagicHaskeller* as the inductive programming core of our approach. The reasons for that decision are several. However, it should be noticed that all the mechanisms for primitive selection can be extended to other systems and solve similar experiments. First of all, *MagicHaskeller* is a general-purpose learning system that works with Haskell, a functional programming language that makes it much easier to add domains and transformations. Besides, *MagicHaskeller* is a very powerful system that can solve many problems using only one example from the data. Finally, with some modification it is also possible to provide *MagicHaskeller* with different domains as different sets of functions. In the next section we will describe *MagicHaskeller's* functionality.

2.4.1 MagicHaskeller

In a nutshell, *MagicHaskeller* is a general-purpose inductive functional programming system that learns Haskell programs from pairs of input-output examples, also expressed in Haskell, using a breadth-first search algorithm and type inference [91]. *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 fx 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 that are defined in its background knowledge to work like the f above. It can take more than one example if they are separated by `&&`[9]. Consider for instance the following two examples of the function `f`:

```
f 9 == 4 && f 25 == 6
```

With this input, *MagicHaskeller* generates a function `f` that computes the square root of the input plus 1. This function can be expressed in Haskell notation as:

```
f = (\a -> 1 + round(sqrt(fromIntegral a)))
```

Note that `f` is composed of five function symbols or constants (`1`, `+`, `round`, `sqrt`, `fromIntegral`). This is what we denote the depth (d) of the solution, the number of function symbols that are combined in it.

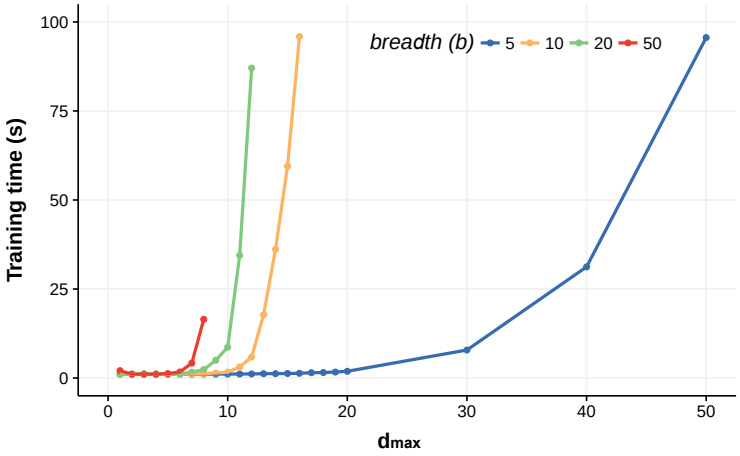
MagicHaskeller works in two steps: *(1)* The *Hypothesis Generation* phase, and *(2)* the *Hypothesis Selection* phase.

[9] *MagicHaskeller* has a web version, where we can see some examples of how it synthesises functions: http://nautilus.cs.miyazaki-u.ac.jp/~skata/MagicHaskeller.html

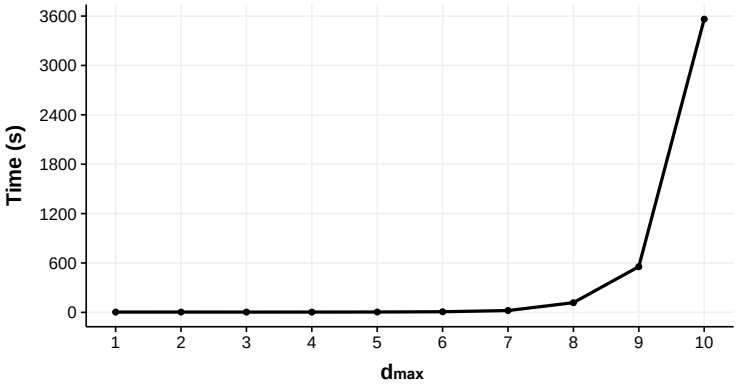
Hypothesis Generation phase: In this phase, *MagicHaskeller* starts with a predefined d_{max} value (maximum depth d allowed for the solution) and a set of b functions in the library. Then, *MagicHaskeller* continues with the preparation of hypotheses by generating all the type-correct expressions that can be expressed by function application and lambda abstraction using up to the maximum depth (d_{max}) using a number of functions from the library that cannot exceed the maximum depth (d_{max}). Although *MagicHaskeller* is very powerful for finding the simplest solutions (that is, those with smallest Kolmogorov complexity), depending on the problem, the solution might require the combination of many function symbols (that is, a solution with a large depth d). When the d required is higher than the d_{max} value used, *MagicHaskeller* is not able to find the solution (because it cannot reach the necessary number of functions to be combined). Trying to increase the d_{max} value to achieve the result may cause an increment of time. On the contrary, trying to reduce d, we may be tempted to add many powerful and abstract functions to the library. But, in this case, *MagicHaskeller* will have too many primitives to choose from (the breadth value b), and may not find it either because of the time needed to combine all of them.

As we mentioned before, it is usually estimated that for the inductive inference, the computational complexity might be in the order of $O(b^d)$ [55]. Figure 2.4 illustrates this by showing the time (in seconds) used by *MagicHaskeller* in this phase when we vary both the number of functions included in the library (b) and the maximum depth value to obtain the solution (d_{max}).

Hypothesis Selection phase: Finally, in this phase we can provide one or more examples (as I/O pairs) to solve a specific problem. *MagicHaskeller* will use the combinations learnt at the previous phase, to find one or more possible solutions to the problem. This solution (if exists) will be a combination of d functions (where $d \leq d_{max}$). In this regard, Figure 2.4(b) shows the time spent during this phase to solve a specific problem (with actual solution of $d = 1$), using the same set of functions (with $b = 15$), but changing the d_{max} value. We acknowledge that d_{max} value has a strong influence too even when there are solutions that require fewer primitives than the maximum depth.



(a)



(b)

Figure 2.4. (a) Hypothesis Generation phase: Time *MagicHaskeller* needs for training with a set of primitives depending on the maximum number of primitives that are allowed in any synthesised function (d_{max}) and the number of primitives in the set (b). In this phase *MagicHaskeller* makes all the possible combinations with the functions included in the background knowledge. Note that this is just the time for combining the functions, here the system is not solving any problem yet. (b) Hypothesis Selection phase: Time *MagicHaskeller* needs for solving the same problem (concatenate two strings), using a set fixed of $b = 15$ primitives, varying d_{max} from 1 to 10. In this phase *MagicHaskeller* tests the combinations generated in a) until it gets all the possible solutions for the problem. In this case, the combinations are already trained, the system is only solving the problem, i.e., applying the functions.

Chapter 3

Data Science Automation

In this chapter we describe the data science process, its phases and its automation. We focus on one of its main stages, data wrangling or data preparation, which is one of the most tedious tasks in any data science project consuming most of the time, but, at same time, it is a essential process before using or analysing the data. The automation of data wrangling could be really helpful in order to reduce time and cost of data-related projects. However, it is not fully automated. Here we present some current researches focused on this problem and its limitations.

This chapter is organised as follows. Section 3.1 presents what data science is and how can be described as a data-driven process. Section 3.2 describes the data science process as a space of activities where several trajectories can occur depending on the project. Then, section 3.4 presents the problem of automating data science, and in particular the data wrangling phase, summarising the current research lines and the future challenges.

3.1 Introduction

The world is changing from 'analog' to 'digital' [79]. This 'datification' is a process of transforming aspects of the world that have never been quantified before into data [84]. Data are collected about anything, at any time, and at any place [166], generating a massive quantity of information about everything. This huge amount of collected data has been called "big data", i.e., massive data sets with large, varied and complex structures. The final use for all these data should be to turn them into real value (improve organisational performance, finding opportunities, making decisions, etc.), even if the variety of structures and formats makes it difficult. However, data science is here to save the day. Data science is the science of data: Its goal is to extract knowledge from all the available data to make informed decisions based on them. Data science tries to answer questions such as "What happened?", "Why did it happen?", "What will happen?", and "What is the best that can happen?" [166].

Data science has gathered public attention in the last years [108]. Yet, the essence of what constitutes data science has been built up for much longer. Data science has deep roots in the history of different academic disciplines. According to [10] there have been four waves of big data that have turned the term "data science" to what we know nowadays: From the first wave in the second half of the twentieth century with the Hubble Space Telescope that captured extremely high-resolution images and scientific data; to the fourth wave currently occurring with the rise of machine-generated data with the Internet of things.

Overall, data science is a field applied both in academia and in industry (*applied data science*) that studies approaches to generate value and insights from

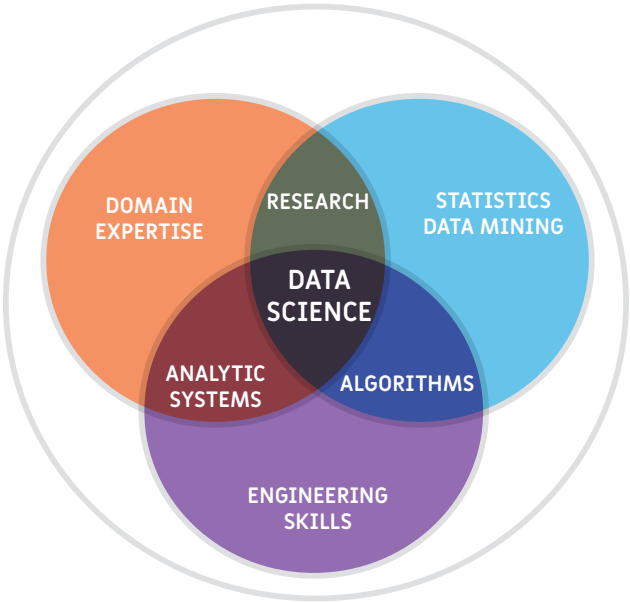


Figure 3.1. Data Science definition by NIST BD-WG [13].

all these data. It is a multidisciplinary field and thus has also a strong connection with other fields such as statistics, machine learning and big data technology [151]. For this, the data scientist needs many different skills and knowledge (see Figure 3.1). Unlike classical data mining methodologies, such as CRISP-DM [14], data science is a data-driven and not a goal-oriented process. CRISP-DM (CRoss-Industry Standard Process for Data Mining) was introduced in 1999 [14]. This methodology was conceived to categorise and describe the common steps in data mining projects, becoming "*de facto* standard for developing data mining and knowledge discovery projects" [111]. CRISP-DM provides an overview of the life cycle of a data mining project: the phases; a set of tasks to be performed; the elements that are produced (outputs); and the elements that are necessary to do (inputs). The cycle consists of six phases (Figure 3.2) and depends on the outcome of each phase which phase or task, has to be performed next. Data mining is not over once a solution is deployed.

These goal-oriented methodologies are focused basically on processes, tasks and roles having the data (that has been already collected) just as a useful element to achieve the main goal. In contrast, in data science the data have a value, and the goal is to find a way to extract this value. This methodology forgets about orders, and becomes more exploratory, following different trajectories depending on the domain and the decisions and discoveries of the data scientist.

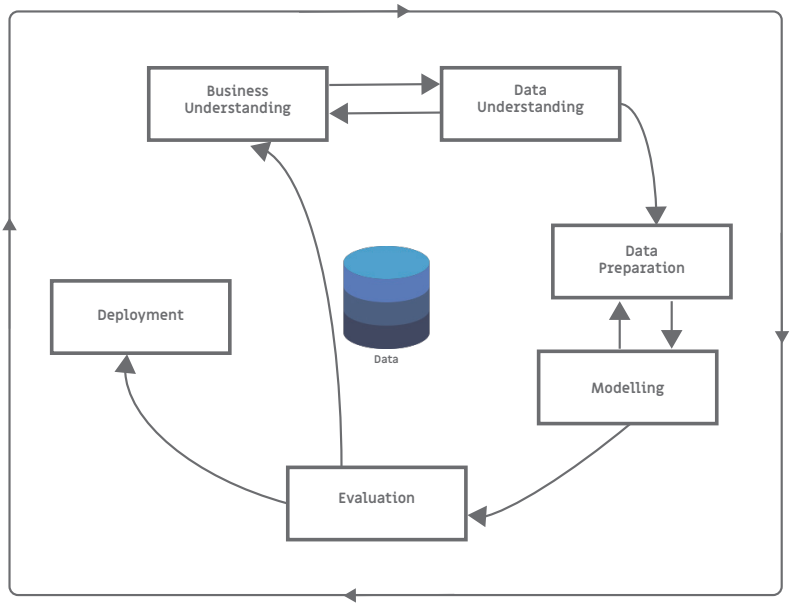


Figure 3.2. Different phases of CRISP-DM life cycle. Adapted from [112].

3.2 Data Science Trajectories

There are many interpretations of data science life cycle and in which way its phases have to be followed. The reality is that, in a data science process, these steps do not follow a fixed or straight path. Instead, data scientists can go forward and back again on their own steps, repeating phases of the process if necessary and skipping some others that are not useful for their project, while exploring the data to find their value. In data science, the order of the activities depends on the data scientist's decisions and discoveries. For instance, when exploring data, anomalous or missing data can be found and thus it may be necessary to go back and do data preparation again (or for the first time); or, when the data to explore is not available, a data source exploration or a data acquisition phase would be in order. It should be noticed that not all the steps have to be done for all the projects nor repeated several times. That is, depending on the project the data science process may change.

Following this idea, different data science projects can follow different trajectories (an acyclic directed graph over activities) through a space like the one shown in Figure 3.3 [114]. In contrast to data mining models and other kind of cycles and processes, there are no arrows here, because a predetermined order to follow through the different activities and phases is not defined. Depending on the project, the next step to take will be decided based

on the available information, including the results of previous activities.

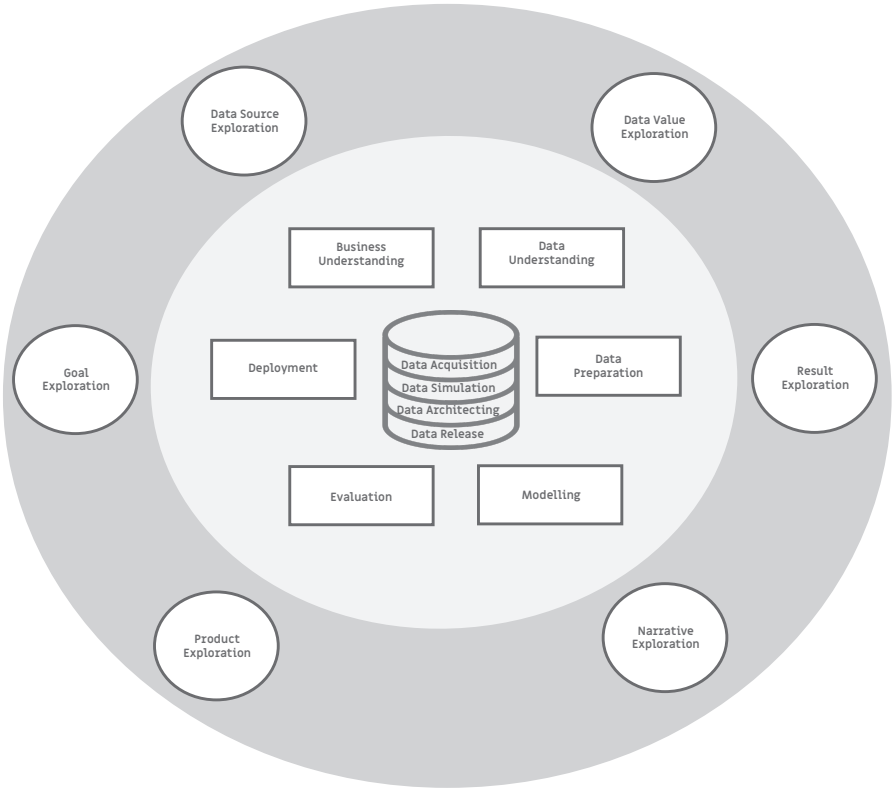


Figure 3.3. Data Science space. Data science projects can follow any path through the different activities included in this space. Adapted from [114].

Note that, in this schema, data are the centre of the process and the activities that are in the middle correspond to those related directly to data management that can answer the following questions:

Data acquisition: how can relevant data be obtained or created, for example by installing sensors or apps?
Data simulation: can we simulate complex systems in order to produce useful data?
Data architecting: how can we design the logical and physical layout of the data to integrating different data sources?
Data release: how can we make the data available?

Surrounding these data phases, we have the classical activities included in a data mining project according to CRISP-DM methodology and, finally, at the

most external part of the space we can find those activities related to extract some value from the data themselves:

Data source exploration: how can new and valuable sources of data be discovered?
Goal exploration: what possible goals can be achieved in a data-driven way?
Product exploration: how can the value extracted from the data be turned into a product?
Data value exploration: what value might be extracted from the data?
Result exploration: how do data science results relate to the goals?
Narrative exploration: what valuable stories (visual or textual) can be extracted from the data?

3.3 Data Wrangling

Following the data science trajectories explained in the previous section, data wrangling (also known as data munging or data engineering) is more than a data preparation phase, being not only a pre-processing step but also requiring some post-processing of the results and some other phases from Figure 3.3, covering all the necessary elements inside the data science process for preparing the data, such as data integration, cleansing or transformation [32]. Data scientists spend (or waste) 80% of the time of their projects preparing unruly and messy data, before it can be explored, since algorithms and tools do not work with messy data [107]. Data wrangling is not an enjoyable job. In fact 57% of the data scientists view cleaning and organising data as the least enjoyable part of their work [146]. The reason for this is simple: data wrangling is still a long and frustrating manual process but, at the same time, a necessary process that cannot be skipped.

Following [133], the data wrangling process can be divided into three high-level groups of problems: data organisation, data quality and feature engineering. Additionally, theses processes may have some sub-processes. Figure 3.4 summarises these groups of processes. Note that this is not a cycle nor a sequential path. As with the data science process, these steps can follow different trajectories or even be skipped.

In the following lines we will try to describe briefly each of these processes and the possible problems or limitations that the data scientists can encounter during their work:

1. **Data Organisation**
    a) **Data Parsing:** identifying the structure of the raw data source.
    $\Rightarrow$ **Possible problems:** wide variety of encoding, multiple tables in a single file.
    b) **Data Dictionary:** understanding the content of the data and translating it into additional metadata (meaning and type of each attribute).

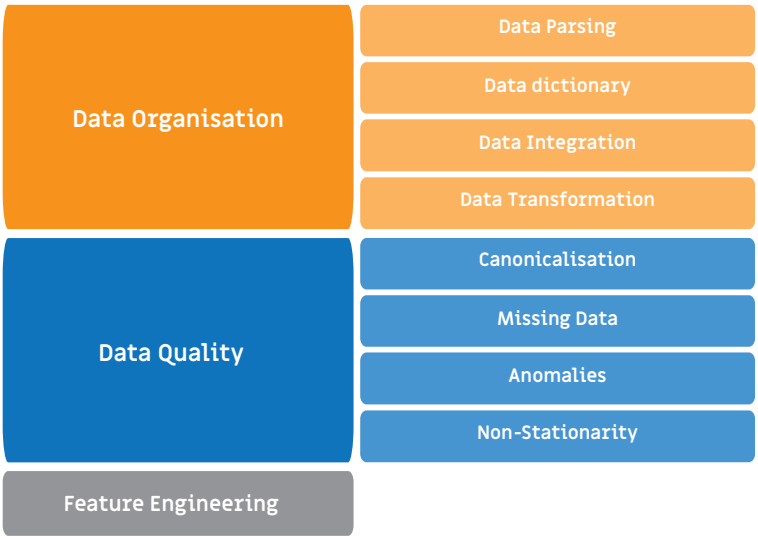


Figure 3.4. Data wrangling classification into different group of problems. Adapted as a summary of the ideas presented in [133].

⇒ **Possible problems:** text documents with a profile, extra headers, additional CSV files, missing or out-of-date metadata.

i. **Table Understanding:** exploring the data to understand their contents, possibly involving interaction with domain experts or the data collector.
ii. **Feature Description:** describing the features (header name or an additional file). When it is not provided, a domain expert should be involved or the information must be inferred.
iii. **Value Understanding:** exploring the values contained in the data.

c) **Data Integration:** combining related information from multiple sources. Aggregating the information in a single data structure.
⇒ **Possible problems:** instalments (monthly/annual updates) separated in different tables.

i. **Record Linkage and Table Joining:** identifying records across multiple tables that correspond to the same entity (entity disambiguation).
ii. **Table Unioning:** aggregating together row-wise different tables containing different entities with the same information.
iii. **Heterogeneous Integration:** combining data from different structured sources (relational tables, time series, etc.) and different physical locations (websites, repositories, etc.).
iv. **Deduplication:** resolving instances containing the same or similar information.

d) **Data Transformation:** transforming the original shape of the data that does not conform to the structure the data analyst needs.
⇒ **Possible problems:** data may not be tabular.
   i. **Table Transformation:** manipulating the data to change their shape (removing rows or columns, switching from wide to tall).
   ii. **Information Extraction:** extracting relevant pieces of information from any kind of structure into multiple features (named entity recognition, relationship extraction, natural language processing (NLP)).

2. **Data Quality**

a) **Canonicalization:** converting entities into a canonical format (common or standard representation).
⇒ **Possible problems:** needs a previous step of identifying which values corresponds to the same entities.
   i. **Cell entity resolution:** resolving instances refering to the same entity.
   ii. **Canonicalization of features:** representing a specific feature type with a standard format
   iii. **Canonicalization of units:** transforming the numerical values and units of a feature into a standard representation.

b) **Missing data:** detecting, understanding and imputing missing values on a dataset.
⇒ **Possible problems:** many machine learning models assume the data are complete.
   i. **Detection:** detecting the missing values.
   ii. **Understanding:** classifying the missing patterns in three groups (MCAR, MAR, MNAR).
   iii. **Repair:** removing or substituting rows or missing values.

c) **Anomalies:** detecting anomalies (supervised, unsupervised, semi-supervised).
   i. **Detection:** detecting anomalies in different ways: Univariate (based only on an individual feature, syntactic or semantic), multivariate (based on multiple features) or statistical (a model is built for the clean distribution of a feature).
   ii. **Repair:** removing or substituting the anomalous entries.

d) **Non-Stationarity:** detecting changes over time (change point detection (CPD) in time series).
⇒ **Possible problems:** dataset shift (distribution differs between training and test), change in how data are collected, units changes, labels recoded.
   i. **Change points:** identifying those points in time with changes in the probabilistic distribution.
   ii. **Protocol changes:** detecting changes on the collection of data over a period of time.

3. **Feature Engineering:** manipulating the data involving changes on the number of features contained in the data or their properties. Some of the most common feature engineering processes are [150]:

    a) **Binning:** reducing the effects of minor observation errors. The original data values in a given small interval are replaced by a value representative of that interval.
    b) **Log Transform:** transforming data distribution more approximate to normal.
    c) **One-hot encoding:** spreading the values in a column to multiple flag columns and assigns 0 or 1 to them. It changes the categorical data to a numerical format.
    d) **Grouping Operations:** grouping the data by the instances so every instance is represented by only one row.
    e) **Feature Split:** extracting the utilisable parts of a column into new features.
    f) **Scaling:** normalising numeric values.

Even when any of these steps could possibly be automated, in this thesis will be focused only on two of them: data transformation and canonicalization. The following sections will be therefore centred in describe the automation of these processes.

3.4 Data Wrangling Automation

During the last few years, we have been seeing a significant increase in the number of tools for automating the modelling phase within the data science process (commonly known as autoML) [67, 83]. However, other phases do not get as much attention even knowing that, as we have seen previously, processes such as data wrangling end up taking a large proportion of the effort dedicated to working with the data. Data wrangling process has not been fully automated, although some of its tasks have been partially automated individually. We have already seen that data wrangling it is divided in very different tasks, so the automation of each of them can vary. In this section we will focus on describing the current lines in this field for automating data wrangling process.

Perhaps, the first question we need to solve here is 'What is automation?'. Automation could be defined as making a process independent from the user, allowing possibly some prior information in the form of examples, or posterior information in the form of feedback. More focused on data wrangling, in an automated system the user should provide information about the domain of application and/or the description of what is required [140]. Automation requires an understanding of what you are looking at, i.e., a context, and for this the background knowledge is specially important. It would be desirable for a system to take more responsibility, if there can be confidence that the system will perform as well as a human expert.

Data wrangling automation is not an easy task. Firstly, data wrangling depends, most of the times, on the domain, type and source of the data, and on the experience and knowledge of the data scientist. Secondly, the process needs posterior feedback that requires manual effort, since the data scientist remains responsible for making many fine grained decisions. However, there is not enough data to automatically learn how to do it. Even when numerous data scientists face data wrangling problems every day, this type of work is not being documented nor the raw datasets published, so there is no (or at least not enough) information about what steps, solutions or intuitions a data scientist actually has to follow when dealing with data and which type of knowledge of each domain, type or source of data is necessary. In [140], Norman Paton exposes this precise problem and encourages data scientists to document the process by describing some use cases in such a way it can be a better known process in the future. However, even if the process were completely documented, data wrangling requires some soft skills (abilities to carry out tasks [117]), such as making decisions, recognising patterns or understanding natural language that are very difficult (most of them impossible) to automate with the current state of AI. Finally, besides the required soft skills, data wrangling is a process that needs many hard skills (technical knowledge) since most of the times data wrangling is a complex task that needs programming skills [140], and/or the use of particular tools (and knowledge on how to use them)[1]. These reasons make data wrangling a process that is very difficult to automate (at least completely). One of the possible solutions for automating data wrangling process, at least for now, can be (and actually is) to automate its more mechanical parts or phases separately. Nevertheless, if enough data wrangling problems were be published, the solution may be summed up in studying how the data scientists behave when they have to deal with these problems[2].

3.4.0.1 Current Research Lines & Future Challenges

The importance of data wrangling in the quality and cost of data science projects has motivated an enormous effort in techniques and tools, including commercial platforms that go beyond ETL tools[3]. For instance, *OpenRefine* [65, 169] provides a set of built-in operators to specify data transformations (assuming the data are in a tabular format). *Ajax* [53] brings a SQL-like language to data transformations. Early work on automating end-to-end data wrangling seems promising, but there is likely much more to do [98].

Some other tools provide learning from examples and some degree of automation for writing partial scripts or patterns to do the transformation automatically. For instance, *Potter's Wheel* [148] infers structures or patterns

[1] Appendix B includes an extended description of the competences and skills identified in data wrangling that can be useful to automate data science process, and data preparation step in particular.

[2] Appendix C includes preliminary ideas for logging data scientists behaviour (these ideas have been published at [26]).

[3] Originally from the data warehousing terminology, ETL is the process responsible for the extraction, transformation and load of the data into a repository.

for data values in terms of a series of default domains (ASCII strings, character strings, integer, sequences of punctuation symbols, ...) and other user-defined domains. Data transformations are graphically specified by the user, except from column splitting for which the user can provide a few examples showing the desired result. Trifacta *Wrangler* [88], based on *Potter's Wheel* generates a ranked list of suggested transformations also inferred automatically from user input, the data type and some heuristics using PbE techniques. Some extensions have been introduced, such as [63], which continually provides suggestions. These systems are able to use different approaches depending on the data type. For instance, they do not treat numbers in the same way as they treat strings. Some of them, such as Wrangler, have specialised "types" for emails, phone numbers, credit cards, social security number and gender. However, these types define that the data must have a particularly-chosen format, but not that the system is able to discover the format or even integrate from different formats. In other words, their pattern generation engines are usually based on predefined rules but not on a fully inductive inference system, able to combine functions as defined by the user (or included in new libraries) to adjust to any possible input-output pairs.

Given this limitation of commercial systems, research has been focused on the inductive inference part of the problem, when the pattern involves the combination of several manipulation functions, a combination that is not part of the original repertoire, so creating new transformations. This generation of approaches is based on *inductive programming* presented in Chapter 2 and recently recognised with a large potential for data wrangling automation [58]. In [29] Cropper et al. investigate the use of the *Metagol* system [131] for learning recursive transformations for unstructured and semi-structured text data. Microsoft also included some of these tools in Excel. One of the reasons of the success of these systems is the use of domain-specific languages (DSLs) [61], which are ad-hoc for data wrangling and data manipulation situations, and reduce the search space considerably.

More recently, Neural Program Induction has been presented as an alternative for learning string transformations. In [138] Parisotto et al. introduce a system that uses Neuro-Symbolic Program Synthesis to learn programs in a DSL by incrementally expanding partial programs. They perform experiments with 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, many not-contemplated examples are impossible to solve. In [155] Shu and Zhang propose NPBE (Neural Programming by Example), a Programming by Example model based on deep neural networks. NPBE induces string manipulation programs based on simple input and output examples 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 for real examples.

A most recent work dealing with automatic data transformation is *TDE* (Transform Data by Example) [68]. In this work, He at al. present 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 50,000 functions and their results tend to have many different solutions that the user has to select which one is the correct one. As we will see in the following chapters, this dependency on the user's manual effort results in worse performance when the domain of the problem is not easy to detect.

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 [36].

Finally, all the systems presented above usually deal with data in tabular format. However, more and more transformations in AI require taking data represented in other format such as matrices (e.g., an image, a dataset, a weight matrix, a result table, etc.) and apply a few primitives (e.g., a convolution, a pivoting, a thresholding, a column-wise mean, etc.). In [171], Wang et al. present an inductive framework called Blaze. The approach employs the abstract semantics of a DSL and a set of examples to find a program whose abstract behaviour covers the examples. The authors used Blaze to build synthesisers for string and matrix transformations. Although the results of their experiments are positive, the number of functions included in the DSL is limited (less than 10 functions). This is motivated by the huge search space, but their very short number of functions dramatically reduces the number of real examples that can be solved by the system.

There are a lot of systems and tools able to automate parts or simple pipelines of data wrangling problems, and, yet, there are a lot of problems that need manual effort and many questions on how to automate them are still in suspense since most of them are made ad-hoc for a particular kind of problem or data. The problem tends to be the information domain that makes the background knowledge a bottleneck in the advance of the field. However, even when all the phases cannot be automated, the efforts on this research field could make the data wrangling process much more bearable and hence data scientists more efficient.

Part III

BK-ADAPT: Automating Data Format Standardisation

Chapter 4

Domain-specific Induction

In the previous chapter we have seen that one of the problems inside data wrangling is converting entities into a common or standard representation. This problem needs a previous step to identify which values correspond to the same domain. As we have said before, given one or two examples, humans are good at detecting the domain of the data and understanding how to solve the problem, because they are able to choose the appropriate solution according to the context. However, the automation of this process could be difficult due to the different existent domains. In this chapter we show that with the use of general-purpose declarative (programming) languages jointly with generic inductive programming systems and the definition of domain-specific knowledge, we are able to detect the domain of the data and solve many specific data wrangling problems related to the standardisation of the values. We apply this approach to several data domains that are automatically solved from very few examples. We also contribute with an integrated benchmark for data wrangling, which we share publicly for the community.

The main contributions of this chapter are:

- The publication of the first data wrangling dataset repository openly available for the data science community.
- The definition of Domain-Specific Induction by structuring the background knowledge in different domains of information.
- A BK-adaptable IP system based on *MagicHaskeller* able to incorporate any type of domain knowledge written in Haskell.

This chapter is organised as follows. Section 4.1 presents the problem of automating data wrangling with an IP system. The problem statement and the domains employed are detailed in section 4.2. The experimental evaluation is included in section 4.3. Finally, section 4.4 remarks the conclusions and future work.

These results have been published in: [18, 23–25].

4.1 Introduction

As presented in chapter 3, the term "data wrangling" usually refers to a great deal of repetitive and very time-consuming data preparation tasks, such as the acquisition, integration, manipulation, cleansing, enrichment and transformation of data from their raw format to a more structured and valuable form for easy access and analysis [87]. The use of ETL tools and other scripting languages for data wrangling partially alleviate the problem, but most of the effort is still

manual and non-systematic. Consequently, progress in the (semi-)automation of data wrangling tasks can have an enormous impact in the costs of data science projects and other data manipulation problems, and can also allow data scientists focus on the valuable knowledge discovery process or on the actual task they are doing.

Many data wrangling problems look automatable, especially because the user can indicate a few illustrative examples that can be used by an IP system [50, 76, 97, 129] to infer a pattern, or inductive hypothesis, that can be applied to complete the rest of the examples automatically. Table 4.1 shows one example that can be completed by non-expert people easily, without further knowledge about the source of the data. It is a very encapsulated problem, inputs and outputs, which should be well handled by machines.

Id	Input	Outputs
1	25-03-74	*25/03/74*
2	29-03-86	*29/03/86*
3	11-02-96	11/02/96
4	11-17-98	17/11/98
5	17-05-17	17/05/17
6	25-08-05	25/08/05
7	30-06-75	30/06/75
8	...	...

Table 4.1. Dataset composed of dates (input) and desired output format. An automatic data wrangling system is fed with the two first examples (in italics) and should automatically complete the rest of the cells (outputs).

Nevertheless, many other data wrangling problems are more challenging, and require an important degree of background knowledge because they depend on the application context of the data. Table 4.2 shows an example of a common data wrangling problem: given a list of dates, extract the name of the month from each of them. The difficulty lies, again, in the different date formats, where the month can be in a different position and these dates delimited by different symbols. In order to understand and complete the transformation, we must know, for instance, that there are only twelve months, that days can only range between 1 and 31 and that years are usually abbreviated with two single digits.

In order to solve this problem, we can split the data wrangling problem into two steps: first, we need to know which the domain is (e.g., dates); and, second, we need to know which transformations we have to apply to the input to obtain the output. For humans this is a relatively easy step because we can recognise the context, but it is not so easy for machines. We need to specify relevant background knowledge depending on the domain. Of course, some of this knowledge may be insufficient to sort out some ambiguities, such as "11.02.18" (this date may be in DDMMYY, YYMMDD or MMDDYY formats). This problem may be automatically solved by computers (through program synthesis)

if they are able to recognise the domain (i.e., dates), and have a sufficiently rich set of functions to deal with the context. Not only does this impose a strong bias that guides the process of finding the transformation pattern that has to be applied but also introduces some useful functions that render the solution (the inferred program) much shorter in terms of the functions involved, i.e., the depth d that we describe in chapter 2.

Of course, dates are not the only kind of data. If we want to deal with physical addresses, we need to provide functions that handle symbols such as "St", "Rd", order of postcodes, etc. Similarly, if we want to deal with people names, we should understand strings such as "Mrs", "Dr", etc. Since all of these cases are very common in databases and other kinds of data that are processed in data science projects, we can add the relevant functions to a general domain library. However, as more kinds of data are required, this library would become huge. Even if the depth would have not changed for the original date problem, the inductive inference process needs to choose from a much larger space of functions, the breath b as we have seen in chapter 2, which makes it much harder. Based on any IP system, which is hypothesis-oriented rather than data-oriented, we see that the effort only depends on these two parameters, d and b, being almost constant on the number of examples. How can we keep both, and especially b, at very low levels?

In this chapter, we propose to control the depth and breadth of the inductive inference problem by choosing a *domain-specific background knowledge* (DSBK) for each kind of problem. The user just needs to suggest which domain to use for a particular problem: dates, times, emails, names, phones, etc. Nevertheless, we envisage that this step is easily automatable too, using some domain inference process that can suggest this to the user, as we discuss at the end of the chapter. It is important to remark that the inductive inference engine is the same, independently of whether we are handling dates or telephone numbers. We do not build a data wrangling system specialised for a particular domain-specific language for each case. Instead of this, we allow the system to use different DSBKs. Thus, in that follows, we will refer to our approach as Domain-Specific Induction (DSI).

Id	Input	Output
1	25-03-74	March
2	03/29/86	March
3	21.02.98	February
4	1998/12/25	December
5	17/05/57	May
6	25-08-05	August
7	06 30 1975	June 8
...	...	

Table 4.2. Dataset composed of dates under very different formats (input) from which the name of the month is extracted (output).

There are several advantages of this approach. The same data wrangling tool can be used for a diversity of problems and domains, without specialised tools for every domain. Second, a set of DSBK libraries can be provided by the tool but also extended by users and communities, especially if the language for adding or modifying functions is general-purpose and well known (e.g., Haskell [141, 142], Prolog [106], etc.).

4.2 Problem Definition

The overall idea is to (semi-)automate the process of transforming data from one format to another, depending on the data domain, using a general-purpose IP system at the core, but enhanced to handle configurable function libraries for each domain (see Figure 4.1). For this, we do the following steps:

1. We take a dataset of input-output pairs and detect the domain of the data.
2. We set the domain by selecting the appropriate background knowledge for the IP system.
3. One or more examples are sent to the IP system as inputs, such as the few first rows in Table 4.1, in the same way a user could complete a few examples. These examples are used for training the system.
4. With the correct DSBK, the system is able to return a list of transformations addressing the problem as the resulting function (f) that is applied to the rest of the inputs, obtaining the new values for the output column.

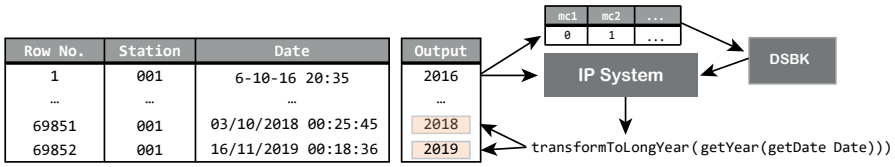


Figure 4.1. Automating data wrangling with inductive programming: process example. The first row (Data and Output) is used as a training example for the IP system. The function returned using the correct domain is applied to the rest of the instances to obtain the outputs.

4.2.1 Domain-Specific Induction

For the purpose of this work we have used *MagicHaskeller* (presented in section 2.4.1) as the IP core system.

By default, *MagicHaskeller* includes a list of 189 basic Haskell functions (the *default* background knowledge). Table 4.3 shows some of these functions[1]. Although *MagicHaskeller* is able to solve many string and Boolean problems by using its default library [90], this list of functions is not enough to solve more complex problems. For instance, the example shown previously in Table 4.1 is impossible to solve with *MagicHaskeller*'s default library since there is a need to replace each dash symbol ('-') with a slash symbol ('/'), and *MagicHaskeller* is unable to generate or use any character or digit if it is not defined as constant in its library or if it is not provided as an input parameter.

Functions
`0 :: Int`
`1 :: Int`
`(++) :: forall a . (->) ([a]) ([a] -> [a])`
`filter :: forall a . (a -> Bool) -> [a] -> [a]`
`isLower :: (->) Char Bool`
`words :: [Char] -> [[Char]]`
`(+) :: Int -> Int`
`True :: Bool`
`False :: Bool`
`isPunctuation :: (->) Char Bool`
`(+) :: (->) Int ((->) Int Int)`
`takeWhile :: forall a . (a -> Bool) -> [a] -> [a]`
`isDigit :: (->) Char Bool`
`not :: (->) Bool Bool`
`(-) :: Int -> Int -> Int`
`(&&) :: (->) Bool ((->) Bool Bool)`
`(||) :: (->) Bool ((->) Bool Bool)`
`not :: (->) Bool Bool`
`(-) :: Int -> Int -> Int`
`reverse :: forall a . [a] -> [a]`

Table 4.3. Some default functions included in *MagicHaskeller*. The complete list of functions is presented in Appendix D.1.

In order to solve this kind of problem we have to add constants to the library and some new functions to work with string problems. For this particular case, we can solve the problem by adding the primitives in Table 4.4 to the library.

Functions	Description
`dash :: [Char]`	Constant for dash ('-') symbol
`slash :: [Char]`	Constant for slash ('/') symbol
`changePunctuationString :: [Char] -> [Char] -> [Char]`	Replace a punctuation sign

Table 4.4. Functions we need to add to *MagicHaskeller* in order to replace a dash symbol with a slash symbol in strings.

Following the example of Table 4.1 and some other examples [134] and the most common operators used by other data science tools [93][88][172], we have added many new functions to *MagicHaskeller* for solving common problems

[1] The complete list of functions included in our approach can be seen at Appendix D.1

related to string manipulation. Concretely, we have added 108 functions to solve the following string operations:

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

With this set of functions (the *freetext* background knowledge) in the system's library, we are able to solve many common string manipulation problems, such as the example in Table 4.1. However, the results can be less accurate for some other examples. Trying to solve the example in Table 1.2 from section 1.1, where we wanted to extract the day from the dates, using the first row as a predicate (`f "25-03-74" == "25"`) the first three results obtained may be: (1) `takeWhile isDigit "25-03-86"`; (2) `getStartToFirstSymbol "25-03-86" dash`; and (3) `take 2 "25-03-86"`. When we apply these functions to the first row, we obtain the desired results, but, what happens if we apply these functions to the rest of the table? We can see the results in Table 4.5. It should be noted that only in the cases when the day is the first element of the date (with solutions 1 and 3) and the next symbol is a dash (with solution 2) the result is correct. The problem here is that we cannot assume that all the data in a column always has the same format. In this case, dates come from very different formats and extracting the first element not always results in getting the day. 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. However, as we have seen in section 2.4.1, it is critical to reduce b while at the same time having the appropriate abstract primitives to learn the function with a short hypothesis (small d). This could be solved by detecting the domain of the data to be

transformed and choosing a domain-specific library for it.

Id	Input	Expected Output	Actual Output (1)	Actual Output (2)	Actual Output (3)
1	*25-03-74*	*25*			
2	03/29/86	29	03	03/29/86	03
3	21.02.98	11	21	21.02.98	21
4	1998/12/25	25	1998	1998/12/25	19
5	17/05/57	17	17	17/05/17	17
6	25-08-05	25	25	25	25
7	06 30 1975	30	06	06 30 1975	06
8	...	...	...	...	...

Table 4.5. Example of a dataset with an input column composed of dates under very different formats, the expected output (day) and the actual outputs obtained using an inductive system with string manipulation functions. The first row is used as input predicate for the system. Green examples are correct results. Red examples are incorrect results. Solution (1): `takeWhile isDigit "input"`; solution (2): `getStartToFirstSymbol "input" dash`; and Solution (3): `take 2 "input"`.

A high number of different domains can appear in any data science project related to data manipulation problems. In order to test the system and as other data wrangling tools have already done [148][161][31], we have selected some of the most used domains [161] and their most common problems [148] to work with. In this sense, for each domain we have a different background knowledge with a set of possible transformations. As we are working with *MagicHaskeller*, they are represented as Haskell functions. These are independent text files, editable by the user, which can be included as a parameter when *MagicHaskeller* is invoked. The DSBK files are:

- *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.

In total we have used 374 different functions. Figure 4.2 shows a summary of the

different background knowledge created. Although we are considering only six domains besides the basic string manipulation functions, it should be noted that many other domains can be created, and it is also easy to build domains that are defined as the union between two existing domains. Also, *MagicHaskeller* can be called with a small d_{max} parameter for one domain to get results quickly and, if unsuccessful, try with a larger d_{max} or another domain. In this way, the search effort can be better handled, depending on the knowledge of the domain and the expected size of the solution.

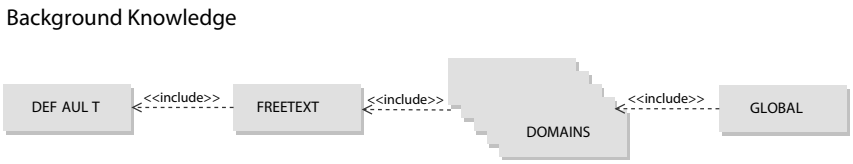


Figure 4.2. The hierarchy of background knowledge's generated.

4.3 Experiments

The aim of our experiments is to analyse the extended capabilities of an IP learning system as a data wrangler. Besides, the experiments explore the improvement when selecting the right DSBK in front of using a general background knowledge or an inappropriate DSBK. Also, and more importantly, we want to compare with other data wrangling systems on a range of data wrangling problems.

To perform the experiments we have followed a trained/test evaluation procedure, similar to [6, 41, 59, 156, 157]. We have used a set of datasets with different data wrangling problems (explained in the following subsection) including inputs and expected outputs. For each of these datasets, we use only the first example as the input predicate for the IP system. Then, we feed the system with this first input/output example using, for each dataset, all the different DSBK. The result is a function f that is applied to the rest of the outputs. The accuracy in each case is the result of comparing the transformed outputs with the real expected outputs.

For replicability reasons, the source code (scripts, domain files, primitive files of *MagicHaskeller*, etc.) of these experiments is available online[2].

4.3.1 Data Wrangling Benchmark

Unfortunately, at the beginning of this work there was no general benchmark or public dataset repository accessible in reusable formats to analyse the quality of new data wrangling tools (for instance, in [41] the Ellis and Gulwani use

[2]Domain-Specific Induction repository: https://github.com/liconoc/DataWrangling-DSI

a dataset with hundreds of data manipulation problems, but the benchmark is not public). In order to overcome this limitation, we collected most of the datasets tested previously in other tools for data manipulation (such as *FlashFill* or *Wrangler*) and presented in the literature [6, 41, 59, 156, 157]. In addition, we generated new datasets based on problems from these papers.

id	Domain	#Ex.	Description of the problem to solve
1	Freetext	12	Complete brackets (From [41])
2	Freetext	12	Extract the first character (From [25])
3	Freetext	24	Delete punctuation (From [41])
4	Freetext	18	Extract the capital letters (From [41])
5	Freetext	12	Extract a substring (From [145])
6	Dates	26	Change the punctuation of a date (From [157])
7	Dates	26	Extract the day from a date (Generated)
8	Dates	12	Extract the day from a date in ordinal format (Generated)
9	Dates	12	Extract the month from dates (Generated)
10	Dates	12	Extract the name of the month from dates (From [145])
11	Dates	9	Add punctuation to a date (From [145])
12	Dates	25	Change date format and punctuation (Generated)
13	Dates	12	Add punctuation and change the format of a date (From [145])
14	Emails	24	Extract words after '@' (From [145])
15	Emails	18	Join words with '@' (From [6])
16	Names	12	Generate a login from a name (Generated)
17	Names	12	Reduce name from one input (From [59])
18	Names	12	Reduce name from two inputs (From [59])
19	Names	12	Extract the honorific forms (From [59])
20	Phones	12	Add phone prefix by country name (From PROSE)
21	Phones	12	Add phone prefix by country name and '+' symbol (Generated)
22	Phones	12	Add a given phone prefix (From [145])
23	Phones	12	Extract a phone number from a string (From [145])
24	Phones	12	Add punctuation to a phone number (Generated)
25	Times	12	Extract the time from a string (Generated)
26	Times	12	Append a specific given time (minutes or seconds) (Generated)
27	Times	12	Increase the hour by a given value (Generated)
28	Times	12	Convert the time to 24h format (Generated)
29	Times	12	Convert time by a given time zone (Generated)
30	Units	12	Extract the units of a value (From [25])
31	Units	12	Detect the system units by the units of a value (Generated)
32	Units	12	Convert a value to a different unit (Generated)

Table 4.6. Datasets included in the new data wrangling repository offered for the data science research community. *#Ex.* shows cardinality. *Freetext* represents the functions created for solving general string manipulation problems.

Overall, we have collected or generated 32 datasets and we have published them on the first data wrangling dataset repository[3]. Table 4.6 shows a summary of the datasets in this new repository.

[3]The Data Wranglig Dataset repository is online and available at: http://dmip.webs.upv.es/datawrangling/

4.3.2 Results

With a focus on our system, Table 4.7 shows the results (accuracy) for all the datasets, using just one example (the first one of each dataset), when *MagicHaskeller* is run without extra DSBK (*default*), adding the string manipulation functions (*freetext*), with a particular DSBK (*dates, emails, names, phones, times, units*) and with all DSBKs together (a unique set of primitives with all the functions together). In each case, *MagicHaskeller* returns a potential solution (or nothing if the problem cannot be solved) which is applied to the rest of input examples to see whether the obtained output matches the expected one. Time execution is limited to 120s with $d_{max} = 4$.

id	Domain	default	freetext	dates	emails	names	phones	times	units	all
1	freetext	0.00	**1.00**	0.00	0.00	0.00	**1.00**	**1.00**	**1.00**	0.00
2	freetext	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.00
3	freetext	0.48	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.00
4	freetext	**1.00**	**1.00**	0.00	0.00	**1.00**	**1.00**	**1.00**	**1.00**	0.00
5	freetext	0.00	**0.55**	0.18	**0.55**	**0.55**	**0.55**	**0.55**	**0.55**	0.00
6	dates	0.00	**1.00**	**1.00**	**1.00**	0.00	**1.00**	**1.00**	**1.00**	0.00
7	dates	0.60	0.60	**1.00**	0.28	0.60	0.60	0.60	0.60	0.00
8	dates	0.00	0.00	**0.91**	0.00	0.00	0.00	0.00	0.00	0.00
9	dates	0.00	0.00	**1.00**	0.00	0.27	0.00	0.00	0.00	0.00
10	dates	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	dates	0.00	**1.00**	**1.00**	**1.00**	0.00	0.00	**1.00**	**1.00**	0.00
12	dates	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	dates	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	emails	0.00	0.04	0.04	**1.00**	0.04	0.04	0.04	0.04	0.00
15	emails	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00
16	names	0.00	0.00	0.00	0.00	**0.91**	0.00	0.00	0.00	0.00
17	names	0.00	0.00	0.00	0.00	**0.91**	0.00	0.00	0.00	0.00
18	names	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00
19	names	0.45	0.73	0.45	0.73	**1.00**	0.73	0.73	0.73	0.00
20	phones	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00
21	phones	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
22	phones	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00
23	phones	0.00	0.27	0.00	0.27	0.27	**1.00**	0.27	0.27	0.00
24	phones	0.00	**1.00**	**1.00**	**1.00**	0.00	**1.00**	0.00	**1.00**	0.00
25	times	0.36	0.91	0.91	0.91	0.91	0.91	**1.00**	0.91	0.00
26	times	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00
27	times	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
28	times	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00	0.00
29	times	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
30	units	0.64	0.18	0.18	0.73	0.18	0.18	0.18	**1.00**	0.00
31	units	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00
32	units	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 4.7. Accuracy obtained per dataset by the DSI approach in each dataset depending on the set of primitives (DSBK) used to train *MagicHaskeller*. The results are obtained with d_{max}=4, $n = 1$ and a maximum execution time of 120s. Maximum accuracy values in bold, where the accuracy is the average of $correct\_examples/(total\_examples - n)$ where $n = 1$.

The results are much better when the right domain is chosen for the problem.

Note that putting all domains together (*all*) implies such big a value of b that *MagicHaskeller* was not able to solve many problems within the maximum time period. In the same way, some specific problems (datasets #10, #12, #13, #21, #27, #29 and #32) cannot be solved using a d_{max}=4 because they need a higher value in order to find the correct solution. It has to be noticed that since all the DSBKs contain some functions for string manipulation, many of them can solve problems related to basic string problems (*freetext* domain). Some problems related to specific domains can also be solved by using basic string manipulation functions, therefore, in this case, any DSBK containing these functions is able to solve the problem. For instance, dataset #6 (*dates* domain) can be solved by using constants and the *freetext* function *changePunctuationString*, as we have seen in section 4.2. Since these functions are included in other domains, not only *dates* has the best accuracy, but also *freetext*, *emails*, *phones*, *times* and *units*.

We have also compared the performance of our DSI approach using *MagicHaskeller* with other data wrangling tools, concretely, *FlashFill* [59]. *Flashfill* works in the same way as our approach, namely, it uses one or more input instances to try to induce a potential solution, which is then applied to the rest of examples. If no solution is found or the problem at hand is not solvable by *FlashFill*, it returns, respectively, a void function or an error.

Table 4.8 shows some illustrative outcomes obtained for each dataset and tool as well as the accuracy values for each dataset. The first instance (in italic) for each dataset (*input* column) is the one used for inducing the solution in the different tools. Here, we can see some strength and weakness in each tool. For instance, *Flashfill* works fine with emails and some basic string transformations, but it fails when it has to deal with titles or honorific forms in people names, with problems related to phones prefixes or times and when it has to work with dates in different formats. For its part, DSI using *MagicHaskeller* is able to find the correct solution for the problem at hand, even with only one example, although it still has problems with unexpected punctuation marks (for instance in dataset #17). In summary, the results show that our approach is able to overcome other tools when dealing with data wrangling problems.

4.4 Conclusions

In this chapter we have adapted a general IP tool to deal with a range of data wrangling problems by using domain-specific background knowledge. Given the impact that the size of the library (b) and the size of the solution (d) have when solving a data manipulation problem, we found a trade-off that produces positive solutions for many datasets. Finding this trade-off and making it work is novel in the context of inductive programming applied for manipulation problems. All this is achieved without the need of increasing the number of examples or using feedback from the user, other than the domain. Users can also edit and create the domain files in a general-purpose functional programming language, making the system more powerful and able to deal with more and more domains. This contrasts with mainstream approaches based on DSLs, where a change

of the DSL aiming at covering other domains cannot be done by the user and might require a redesign of the system. Furthermore, the experiments show that our DSI-based approach gets better results than DSL-based approaches, such as FlashFill, mainly due to its adaptability to the problem domain by using domain-specific background knowledge (DSBK).

This shows that for these repetitive snippets of code that are necessary for data manipulation problems, we can replace some of the tedious programming effort by the selection of libraries or the definitions of proper functions to handle existing or new domains. Functional programming languages, as we have seen, are particularly appropriate for this. In the end, these data wrangling systems over functional programming languages can actually have the effect of truly incorporating automated programming and program synthesis as a toolbox, even if at the level of the generation of small snippets, for these kinds of applications.

Finally, we provide what might be considered as the first public repository of datasets for testing data wrangling tools. Although there are several approaches and systems in the literature dealing with the issue under consideration here, none of them provide public access, nor a complete description of the datasets used for their evaluation. In this way, the evaluation procedures are not replicable and neither is the data reusable. We have collected different problems from the literature and related software, together with a few freshly-generated ones. With all these data, we have generated a variety of datasets for six different domains covering different specific problems in each of them. This repository is open and freely available, and it is already being extended with more types of problems and domains.

In the following chapter, we will show how to automate the detection of the domain at hand by using machine learning techniques. The idea is to learn a meta-model that is able to automatically select (or suggest) the appropriate DSBK from the features of the problem.

id	input	expected output	FlashFill	DSI
3	*1-452-789-4567*	*14527894567*		
	1-406-789-1562	14067891562	14067891562	14067891562
	1-4565	14565	14565	14565
	Etiam dapibus.	Etiamdapibus		Etiamdapibus
		Accuracy:	0.48	1
4	*International Business Machines*	*IBM*		
	Principles Of Programming Languages	POPL	POPL	POPL
	International Conference on Data Mining series	ICDM	ICDM	ICDM
	Association of Computational Linguistics	ACL	ACL	ACL
		Accuracy:	1	1
8	*3/29/86*	*29th*		
	10 12 69	10th	12th	10th
	04/05/99	04th	05th	04th
	27/07/2007	27th	07th	27th
		Accuracy:	0	1
9	*2 of September of 2010, Monday*	*September*		
	13 November 2008	November	2008	November
	Tuesday, September 16, 1986	September	September	September
	February 4, 2008	February	2008	February
		Accuracy:	0.36	1
14	*Nancy.FreeHafer@fourthcoffee.com*	*fourthcoffee.com*		
	iabetrae@yahoo.es	yahoo.es	yahoo.es	yahoo.es
	Sb.edhxo.sk8@hotmail.com	hotmail.com	hotmail.com	hotmail.com
	dala_aguera_m500@hotmail.com	hotmail.com	hotmail.com	hotmail.com
		Accuracy:	1	1
15	*Sophia & domain*	*Sophia@domain.com*		
	elizabeth & gmail	elizabeth@gmail.com	elizabeth@gmail.com	elizabeth@gmail.com
	joypao & hotmail	joypao@hotmail.com	joypao@hotmail.com	joypao@hotmail.com
	casper & canal13	casper@canal13.com	casper@canal13.com	casper@canal13.com
		Accuracy:	1	1
17	*Damian Gobbee*	*D.Gobbee*		
	Antonio Hege	A.Hege	A.Hege	A.Hege
	Damancio Hivser-Kleiner	D.Hivser-Kleiner	D.Kleiner	D.Hivser-Kleiner
	Prof. Edward Davis	E.Davis	P.Davis	E.Davis
		Accuracy:	0.63	0.91
19	*Dr. B. Schdur*	*Dr.*		
	Prof. H. Huifen	Prof.	Prof.	Prof.
	Louis Johnson, PhD	PhD	Lou	PhD
	Robert Mills		Rob	
		Accuracy:	0.72	1
20	*235-7654 & Taiwan*	*(886) 235-7654*		
	17-455-81-39 & Spain	(34) 17-455-81-39	(886) 17-455-81-39	(34) 17-455-81-39
	618-4390 & Panama	(507) 618-4390	(886) 618-4390	(507) 618-4390
	25-613-24-50 & Chile	(56) 25-613-24-50	(886) 25-613-24-50	(56) 25-613-24-50
		Accuracy:	0	1
23	*23/11/18 425-785-4210*	*425-785-4210*		
	425-613-2450 000-000	425-613-2450	2450 000-000	425-613-2450
	[TS]865-000-0000 - 06-23-09	865-000-0000	06-23-2009	865-000-0000
	17:58-19:29, 425-743-1650	425-743-1650	425-743-1650	425-743-1650
		Accuracy:	0.36	1
25	*08:55 PM CET*	*08:55*		
	20:15:00	20:15:00	20:15:00	20:15:00
	10:05:00 AM	10:05:00	10:05:00	10:05:00
	UTC 21:20	21:20	UTC 21:20	21:20
		Accuracy:	0.91	1
28	*01:34:00 & 5*	*06:34:00*		
	01:55 & 5	06:55	06:55	06:55
	16:15:12 & 5	21:15:12	06:15:12	21:15:12
	21:20 & 5	02:20	06:20	02:20
		Accuracy:	0.10	1
30	*56.77cl*	*cl*		
	84Kg	Kg	Kg	Kg
	39.88 A	A	A	A
	1nm	nm	nm	nm
		Accuracy:	1	1
31	*56.77cl*	*Volume*		
	84Kg	Mass	Volume	Mass
	39.88 A	Electricity	Volume	Electricity
	1nm	Length	Volume	Length
		Accuracy:	0.10	1

Table 4.8. Example of results obtained with DSI (using *MagicHaskeller* as IP core) compared with *FlashFill*. *Output* is the expected output. The first row of each dataset (*id*) is the example given to *FlashFill* and *MagicHaskeller* to learn. Green and Red colours mean, respectively, correct and incorrect results. The accuracy is the percentage of correct examples.

Chapter 5

Dynamic Background Knowledge

In the previous chapter we have adapted a general IP tool to deal with some data wrangling problems by using domain-specific background knowledge. However, the approach presented gets a semi-automatic system for data feature transformation since the user still needs to select the appropriate background knowledge. In this chapter we help alleviate this problem by using the inductive programming system presented in the previous chapter, updated with a dynamic background knowledge (BK) fuelled by a machine learning meta-model that automatically 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 extended benchmark for data wrangling, that we share publicly for the community.

The main contributions of this chapter are:

- A method to describe the domains and the problems by extracting meta-features from the examples.
- A domain-classifier and a function-ranking able to use the meta-features to detect the domain of the data and the particular problem to solve (the functions needed).
- The definition of a dynamic background knowledge, able to adapt the background knowledge depending on the input examples, their domain and the problem to solve.
- An update of the system presented in the previous chapter, BK-ADAPT, that uses dynamic background knowledge to correctly solve data wrangling problems independently of their domain.

This chapter is organised as follows. Section 5.1 addresses the problem of selecting the domain to be passed to an IP system and section 5.2 summarises the changes compared with the previous version. Section 5.3 describes the new approach for handling the background knowledge. The experimental evaluation is included in section 5.4. Finally, section 5.5 closes the chapter with the conclusions and future work.

Parts of this chapter have been published in: [17, 21, 22].

5.1 Introduction

Data science must integrate data from very different data sources (e.g., databases, repositories, webs, spreadsheets, documents, etc.). Rarely does these data come in a clean, consistent and well-structured way, imposing a non-negligible manual

effort. As we have seen in the previous chapter, Inductive Programming can be successfully applied to data wrangling problems using declarative background knowledge. 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 order to automate this process, the system (1) must recognise without human help 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, while (3) keeping the depth (d), and especially the breath (b), at very low levels.

We have seen that we can adapt a general IP tool to deal with data wrangling problems by using domain-specific background knowledge. With the system created in the previous chapter we can make some tasks easier and faster just by selecting the right domain of the data and incorporating automated program synthesis for these kinds of applications. Even so, in the version presented in the previous chapter, we need the user to provide the right domain of the data, making the system semi-automatic.

In this chapter, we propose to automate data feature transformation in such a way the user is no longer required to provide the domain of the data, while controlling 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 background knowledge into specific subsets (domains) and select the most appropriate one (using the domains of the previous chapter, but automating the process of domain selection). Second, we build a ranker that selects the most appropriate primitives depending on the problem. In both cases we use off-the-shelf Machine Learning (ML) techniques applied to a set of meta-features 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 new datasets to the data wrangling benchmark.

5.2 Upgraded Approach

As we have presented in the previous chapter, we propose to use a general-purpose IP system provided with a suitable set of primitives as background knowledge. In this case, the selection of this background knowledge will be done automatically by the system. Hence, in this upgraded approach, the automation of data manipulation tasks is done as follows:

- We take one example which is used to automatically select the appropriate domain and/or the set of primitives that form the dynamic background knowledge automatically.
- One or more examples are sent to an IP system.

- Using the selected background knowledge and the examples, the IP system learns a function `f` (if exists) that correctly transforms the input of the examples to the given outputs.
- The function `f` is applied to the rest of the inputs, obtaining the new values for the output column automatically.

We use the term *global* for the set of all primitives, including *default*, *freetext* and all the domain functions. Of course, using this massive background knowledge 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 previous chapter. 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 background knowledge.

5.3 Method

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 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 (e.g., *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 5.1 shows an example of some of these characteristics extracted from dates and emails.

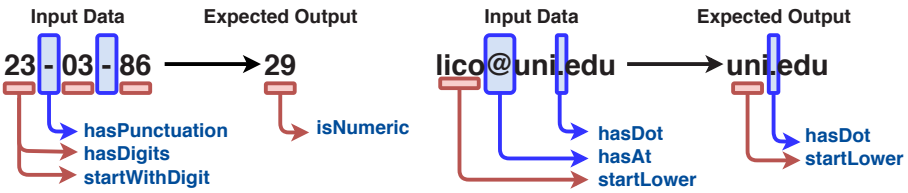


Figure 5.1. An example of meta-features that can be extracted from the examples of different domains (dates and emails in the figure).

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 background knowledge? As explained in the following paragraphs and illustrated

in Figure 5.2, 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 background knowledge 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 background knowledge 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.

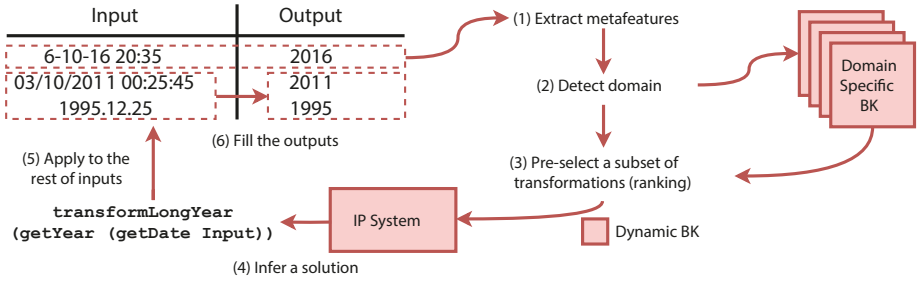


Figure 5.2. Automating data wrangling with IP: process example. The first row (Input and Output) is used as an input example for the IP system. The function returned is applied to the rest of the instances to obtain the outputs.

5.4 Experiments

In addition to the benchmark presented at the previous chapter, we generated 95 new examples for all the domains (with 6 instances each) with different data wrangling problems including names, phones, emails, times and unit transformations. All the datasets are published at the first *data wrangling dataset repository* presented in the previous chapter[1] and are summarised in Table 5.1.

id	Description	Expected Output
1, 2	Add punctuation	The date 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 created with name and 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 using time zone
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

Table 5.1. Datasets included in the new data wrangling repository offered for the research community.

[1]The repository is available at: http://dmip.webs.upv.es/datawrangling/index.html

In this section, we present a summary of the results obtained by applying our system and other related systems on this repository[2].

5.4.1 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.

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). 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 and kappa for the domain classifier, and *AUC* (the Area under the ROC curve) for the primitive estimator. For each *domain classifier* and *primitive estimator* trained, we also applied a leave-one-out cross validation to the induction problem, such that, for each fold from the six examples that the test dataset contains, 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.

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 5.2). Analogously, the experiments performed with the *primitive estimator* (see Appendix D.3 for more details) obtained an average AUC=0.97, showing that it can predict accurately the functions needed to solve the problems.

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**

Table 5.2. Results for the domain detection using the meta-features with different machine learning methods. The best results are highlighted in bold.

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.

[2]The complete results of these experiments can be found in Appendix D.3 and the code is available at: https://github.com/liconoc/DataWrangling-DSI

4. *User Domain*: We know (or the user gives) the correct domain (DSBK) for the problem (as we do in our previous approach presented in Chapter 4).

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 *default*, *freetext* and *global* as baselines since they do not constitute any improvement in the handling of the BK. Strategy *user domain* 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 *inferred domain*, *ranking* and *inferred domain + ranking* introduced in section 5.3) are able to be on a par in accuracy with a human reference and improve the performance over the baselines in time by reducing the hypothesis space. Accuracy is computed as the ratio of correctly covered examples by the induced pattern.

In the case of the strategies using the ranking of functions, we order the functions included in the *global* library (in the case of the *ranking* strategy) or in the DSBK corresponding to the detected domain (in the case of the *inferred domain + ranking* strategy), by the probabilities returned by the *primitive estimator*. Then, we create the dynamic background knowledge by adding one function at a time, starting from the most probable one, until a result is found or all the functions in the library are added. We call b_{max} the size of the dynamic background knowledge generated (the final number of functions included).

Figure 5.3 shows the average time for the seven strategies, that includes the duration of the whole process: from the extraction of the first example to the automatic transformation of the rest of the outputs (as described in Figure 5.2). Concretely, we have measured: (1) time for detecting the domain (strategies *inferred domain* and *inferred domain + ranking*); (2) time for ranking the functions (strategies *ranking* and *inferred Domain + ranking*); and (3) time of running the IP system (all strategies). Considering the running times, we can conclude that the proposed strategies are able to speed up the whole process, even when they need to detect the domain and rank the functions.

If we consider accuracy, Figure 5.4 shows the average for the seven strategies. We can see that 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 *ranking* and *inferred domain + ranking* are on the same level as strategy *user Domain*, which requires a human. This means that we are able to automate the process obtaining the result just as a human can obtain it, but reducing the time needed to do it. We state that the results using the *ranking* strategy lead to significant results and, to

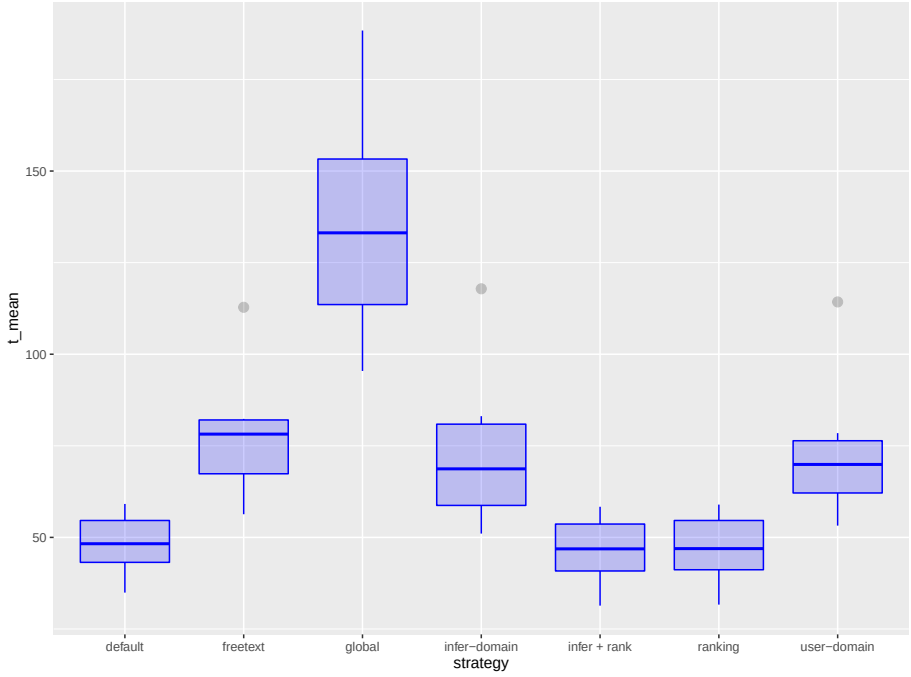


Figure 5.3. Average of time needed to find the solution depending on the strategy used.

prove it, we have run a significance [69][3]. The absolute value of the test statistic for our results, 1.1005, is lower than the critical value of 1.6528, so we reject the null hypothesis and conclude that the results using the ranking of functions are better at the 0.05 significance level.

In the case of the *inferred Domain* strategy the difference in the results compared to the *user-domain* strategy, is the misclassification of the emails domain, which means the strategy is using an incorrect domain and the right solution is not obtained in this case.

Figure 5.5 illustrates the size of the dynamic background knowledge, comparing the strategies using the ranking (*ranking* and *inferred domain + ranking*), depending on the domain of the problems. Here we can see that ranking only the functions included in the detected domain reduces significantly the background knowledge to an average of eight functions. Notice that, as we have seen in chapter 4, the number of functions included in each DSBK is: 162 for *dates*; 162 for *emails*; 102 for *names*; 116 for *phones*; 129 for *times*; 148 for *units*; and 124 for *freetext*. Besides, *global* includes all the functions together. So, we can see that, in general, the ranking of functions is able to drastically reduce

[3]We have used the implementation of the Two-Sample t-Test from the *stats* package from the R language.

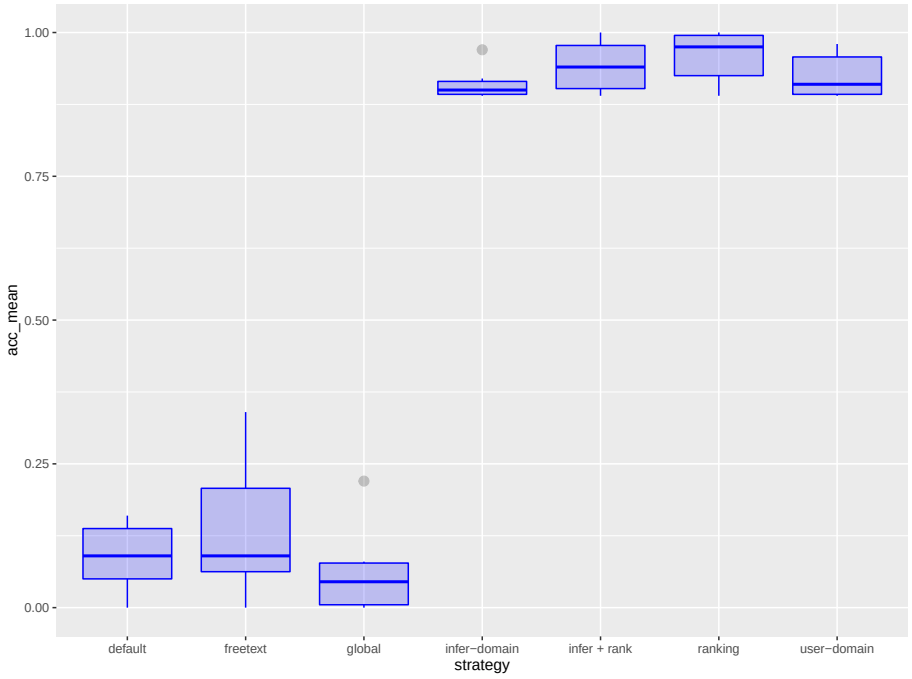


Figure 5.4. Average of accuracy obtained by each strategy.

the number of functions needed, adapting the background knowledge according to the problem to solve.

5.4.2 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 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.

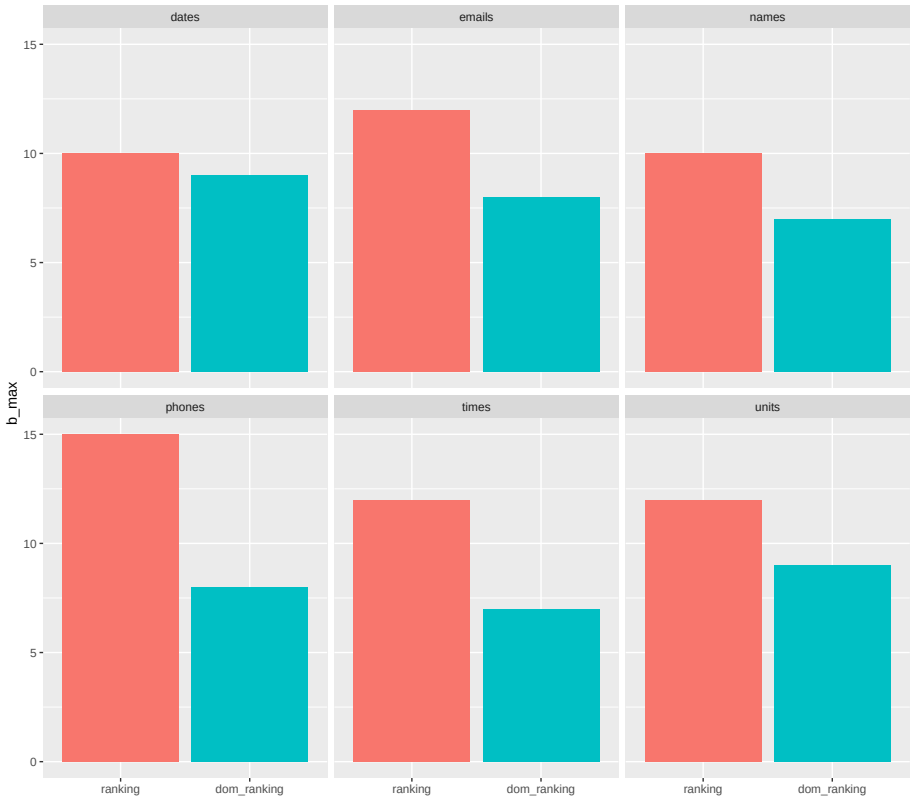


Figure 5.5. Average of size of the dynamic background knowledge using the ranking of functions and the ranking with the inferred domain, for the six domains analysed in this chapter. We omit here the other strategies since the number of functions in their BKs is always the same.

Table 5.3 shows some illustrative outcomes obtained by the analysed systems for some datasets 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 Appendix D.3.

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 that *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[4] that can also be used in order to test data transformation systems.

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

input	output	FlashFill	Wrangler	TDE	DBK
03/29/86	*29*				
74-03-31	31	03	03	*31*	31
99/12/13	13	12	12	/1	13
11.02.96	11	02		/1	11
31/05/17	31	05	05	31	31
25-08-85	25	08	08	25	25
	Accuracy:	0	0	0.5	1
Dr. Eran Yahav	*Yahav, E.*				
Prof. Kath S. Fish	Fish, K.	Fish, Kath S.	S, K.	*Fish, K.*	Fish, K.
Bill Gates, Sr.	Gates, B.	Sr., G.	Sr, G.	Sr.	Gates, B.
George Ciprian Nec	Nec, G.	Nec, C.	Nec, C.	Nec	Nec, G.
Ken McMillan, II	McMillan, K.	II, M.	II, M.	II	McMillan, K.
Mr. David Jones	Jones, D.	Jones, D.	Jones, D.	Jones, D.	Jones, D.
	Accuracy:	0.2	0.2	0.25	1
1:34:00 PM CST	*1:34:00*				
01:55	01:55	01:55	01:55	*01:55*	01:55
3:40 AM	3:40	3:40	3:40	h3:40 A:00	3:40
07:05:59	07:05:59	07:05:59	07:05:59	h7:05:5:59	07:05:59
08:40 UTC	08:40	08:40	08:40	r8:40 U	08:40
16:15:12	16:15:12	16:15:12	16:15:12	h6:15:1:12	16:15:12
	Accuracy:	1	1	0	1
1441.8mg → g	*1.4418001*				
84kg → *g*	84000.0	8.4418001		*84000.0*	84000.0
14300ms → *s*	8700000.0	1.4418001		s	14.3
87 s → *ns*	8700000.0	8.4418001		ns	8700000.0
12.20dg → *mg*	1220.0	1.4418001		mg	1220.0
1854 dam → *dm*	185400.0	1.4418001		dm	185400.0
	Accuracy:	0	0	0	1

Table 5.3. 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.

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 [138], 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 be beaten by future variants of these and other approaches.

5.5 Conclusions

Most data science applications require the manipulation of data that are 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 chapter, 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 95 new 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) inductive programming and machine learning 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.

Part IV

AUTOMAT[R]IX: Automating Matrix Transformations

Chapter 6

Learning Simple Matrix Pipelines

In the previous part we saw how to automate the transformation of strings based on the characteristics of the problem. As said in the Section 1.1 one process repeated in many data-related projects that is difficult to automate is programming. In this chapter, we present AUTOMAT[R]IX that, as far as we know, is the first system able to automatically synthesise programs in R[1] given an input data matrix, a partial output matrix filled by the user representing the expected solution and, optionally, a brief description in natural language of the desired result or the operation to perform. The goal of this approach is dual: first, we want to extend the range of automated data transformation, not only to strings, but also to complex structures, such as matrices; and second, we want to automate the generation of programming code related to data science. We assume that there is a set of functions in R that can be applied to a matrix in order to obtain the result, and there is a user that needs some assistance to generate the code automatically only using a few examples from the result. The learning algorithm is able to induce the correct matrix pipeline transformation by composing primitives from a background knowledge. Because of the combinatorics of primitives and operations for generating possible transformations —exponential on the size of the library and the number of primitives to be combined—, we need to use the characteristics of the input and output matrices, and the primitives themselves, in the form of constraints to speed up the process. After that, a probabilistic model estimates the probability of each sequence of primitives from primitive use frequencies and documentation and completes the output matrix, automatically producing the R code, ready to be inserted into the data science pipeline.

The main contributions of this chapter are:

- A new probabilistic algorithm that dynamically adapts the background knowledge based on prior information about the use of the functions by the programming community and tips provided by the user in natural language describing the problem to solve.
- The first IP system able to induce R functions based on examples of partially-filled matrices and tips from the user.

This chapter is organised as follows. Section 6.1 presents the problem that programming can imply to many data scientists and how can be automated. Section 6.2 defines the problem that we address in this thesis. Section 6.3 gives details of our approach and how it leverages novel and traditional ideas in artificial intelligence to solve this new problem effectively and efficiently,

[1] https://www.r-project.org/

including the algorithm that is constrained by the matrix dimensions and uses probabilities estimated from the textual hints. Section 6.4 includes experiments with artificial and real data. Finally, section 6.5 closes the chapter with the applicability of the system and the future work.

These results have been published in: [19, 20].

6.1 Introduction

Many areas in artificial intelligence, from data pre-processing to optimisation algorithms, from image transformation to the visualisation of tables and results, require common operations using matrices. This is especially the case in machine learning and data science, where programming languages, such as R or Python, are the most used languages to manipulate data [27]. While libraries and hand-crafted algorithms usually capture the core data pipelines, there is an extensive use of glue code, small snippets that perform simple transformation between the output of one module to the input of the following one. Sometimes, these transformations must be done by experts without a profound programming knowledge, with frustrating results [85].

Matrices are a very common way of working with data. Data-frames, tables, spreadsheets, bi-dimensional arrays and tensors are similar structures that can be assimilated to matrices. Matrix algebra can be applied to transform data or extract a variety of useful information. It is a common strategy for programmers, AI experts and data scientists to use an example of a problem and use it to solve the desired transformation or detect where it fails. A few examples are used to 'verify' whether the matrix snippet is doing the right processing, before applying the transformation to other examples.

For instance, consider an AI expert or a data scientist who wants to extract the positions of the non-empty values in the data matrix shown in Figure 6.1a. She may just figure out the output she is expecting as a result of the desired transformation (represented in Table 6.1b), which may also be used to check the snippet once it is written. Interestingly, as happens with humans, having both the input and output matrices could be sufficient for *an automated system* to learn this transformation smoothly. This is what we do in this chapter.

NA	0.30	0.50	NA	NA	NA	NA
NA	NA	NA	0.90	NA	NA	0.40
NA	NA	NA	NA	NA	NA	NA
NA	NA	NA	NA	0.60	NA	NA
NA	NA	NA	NA	NA	NA	NA

(a) Matrix A with some NA values.

1	2
1	3
2	4
4	5
2	7

(b) Position (row, column) of non-NA values in A.

Figure 6.1. Example of data transformation using matrices. The snippet must transform the matrix on the left into the matrix on the right. Can you code it?

Common strategies to solve this problem by hand would be to program a snippet using a traditional loop, think of a more algebraic function that avoids the loop or simply check Stack Overflow[2] to find an elegant transformation. Instead, we could rely on a system that takes Tables 6.1a and 6.1b as inputs, and generate an elegant code snippet in the programming language *R* such as `which(!is.na(A), arr.ind=TRUE)`? Note that in this example there is no similarity whatsoever between input and output. The input contains real numbers and `NA`s, and the output only has integer numbers, none of them in common with the input. Also, the dimension of the input matrix is 5×7, while the output matrix has dimension 5×2, where the same number of rows is just a coincidence. Is this problem solvable at all? And, to make it more challenging and realistic, what if we only give some of the rows (or even a few cells) of the solution?

As we have seen in chapter 2, generating code from input-output pairs falls under the area of programming by example (PbE) in program synthesis, and more generally inductive programming [62]. From the perspective of inductive programming, matrix transformation has several characteristics that make it more feasible than other problems: only one data structure is considered (matrices), we can use the matrix dimension as a constraint to restrict the search, and functional programming (with higher-order functions such as `apply`) is particularly appropriate here. A first key insight to tackle our problem is that very interesting and complex transformations can be obtained by composing very few primitives. A second observation is that the primitives used in matrix pipelines usually have very low arity, with many of them having just one argument. A third consideration is that the input and output matrices represent just *one* —albeit rich— example: the information to exclude the infinitely-many alternative transformations must come from the (partial) content of the matrices and a strong simplicity prior. This makes other approaches that require significant amounts of data unfeasible and suggests a learning approach that is based on a compositional search.

In this chapter, we present a system that is able to learn simple matrix pipelines from: (1) *one* input data matrix, (2) *one* partial output matrix filled by the user representing the desired transformation, and (3) *optionally*, a short description or hint in natural language of the desired transformation. The system works with a library of b primitives to combine into a snippet of at most d primitives. Because the combinatorics of primitives and operations is in the order of b^d as we discussed in chapter 2, we use several strategies to reduce the search space. First, we use the characteristics of the input and output matrices, and the primitives themselves, in the form of constraints. Basically, our system checks that the sequence of compositions is consistent with the dimensions. Second, we estimate the a priori probabilities of the primitives according to how frequent they are used on Github. Third, when available, we can use small hints in natural language given by the user, such as a short text of the form: "positions of non-empty values" (see Figure 6.1). This helps us estimate the

[2] https://stackoverflow.com/

conditional probabilities for primitive sequences. We use all this information for a tree-search procedure that re-estimates branching candidates dynamically.

6.2 Problem Definition

We assume there is a set of operations that can be combined and applied to a matrix A in order to obtain a result S. The operations are primitives or functions in some specific language (in our case, R) and a human needs some assistance to generate the code from a single example. This is the setting that serves as problem formulation:

1. We are given an input matrix A: a finite real matrix of size $m \times n$ $(m, n > 0)$.
2. We are given a partially filled matrix B: a finite real matrix of size $m' \times n'$ $(m', n' > 0)$ where only some elements are filled and the rest are empty (we will use the notation '$\cdot$').
3. Optionally, we are given some textual hint or short description T in natural language: this is provided by the user, describing the problem to solve.
4. We look for a function $\hat{f}$ such that $\hat{f}(A) = S$, where S is a finite real matrix of size $m' \times n'$, such that for every non-empty $b_{ij} \in B$ the corresponding $s_{ij} \in S$ matches, i.e., $b_{ij} = s_{ij}$.
5. We produce the function $\hat{f}$, expressed as a composition of matrix operations in a given programming language.

As additional criterion we will consider that the representation of $\hat{f}$ in the programming language should be as short as possible in number of functions combined , and we will also allow for some precision error ϵ (so that we relax item 4 above with $|b_{ij} - s_{ij}| \leq \epsilon$, instead of $b_{ij} = s_{ij}$). We use the notation $\hat{f}(A) \models_\epsilon B$ to represent this, and say that the transformation *covers* B.

As a basic example, consider the matrices A and B:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 4 & 2 & 6 \\ 3 & 8 & 7 \end{bmatrix}$$

$$B = \begin{bmatrix} 8 & 13 & \cdot \end{bmatrix}$$

where A is the input matrix and B the partially-filled output. We try to find $\hat{f}$ such that $\hat{f}(A) \models_\epsilon B$. In this case the function `colSums` in R, which adds the values columnwise, gives the following matrix S that covers B.

$$S = \begin{bmatrix} 8 & 13 & 18 \end{bmatrix}$$

Note that we look for a system that: (1) works with only *one* input matrix and only *one* partially-filled output matrix, and nothing more (the textual hint is

optional), (2) automatically synthesises the composition of primitives in the base programming language that solves the above problem, and (3) returns the complete transformed matrix and the synthesised code.

As far as we know from the related work seen in the previous section, no other approach is able to solve this problem using only one or few cells of the solution matrix.

6.3 Method

We emphasise a series of characteristics of this problem: (1) the combination of a few functional primitives can achieve very complex transformations, (2) the arity of the primitives is usually low (one in many cases), so the snippet becomes a pipeline, where the output of one primitive becomes the input of the next one, (3) we work from one example and (4) we want the shortest transformation with the given primitives, as built-in primitives usually lead to more efficient transformations. All these characteristics suggest that the problem can be addressed by exploring all combinations of primitive sequences, by using a strong simplicity bias (the number of primitives used). This strategy is common in other inductive programming scenarios [92, 119, 120, 123] but it must always be coupled with some constraints (e.g., types, schemata, etc.) or strong heuristics. In our case, we will use the dimensions of the matrices as the main constraint for reducing the combinatorial explosion, as well as some priors about the frequency of each primitive and, optionally, some posteriors using text hints in order to guide a tree-based search where each combination of functions will be sorted and selected based on its assigned probability.

6.3.1 Dimensional constraints

We consider the background knowledge as a set of primitives G. This number of primitives $|G|$ taken into account for the search is known as the *breadth* (b) of the problem, while the minimum number of such primitives that have to be combined in one solution is known as *depth* (d). Clearly, both depth and breadth highly influence the hardness of the problem, in a way that is usually exponential, $O(b^d)$ [45] affecting the time and resources needed to find the right solution. This expression is exact if we consider unary primitives, so that solutions become matrix operation *pipelines*, i.e., a string of primitive $c = g_1 g_2 \dots g_d$.

The first optimisation to this search comes from the constraints about the dimensions of the primitives and the input/output matrices. For each matrix primitive g we take into account the dimension of the input and output at any point of the composition, and also some other constraints about minimum dimension (for instance, calculating correlations, function `cor`, requires at least two rows, i.e., $m > 1$). More formally, for each primitive g we define a tuple $\langle m_{min}, n_{min}, \tau \rangle$ where m_{min} and n_{min} are the minimum number of rows and columns (respectively) for the input (by default $m_{min} = 1$ and $n_{min} = 1$), and $\tau : \mathbb{N}^2 \to \mathbb{N}^2$ is a *type function*, which maps the dimension of the input matrix

to the dimension of the output matrix. For instance, for $g =$`colSums`, $m_{min} = 1$ and $n_{min} = 1$ as you need at least one column and row for the primitive to work. $\tau(m, n) = (1, n)$, because g takes a matrix of size $m \times n$ and returns a matrix of size $1 \times n$. Similarly, for $g =$`cor`, $m_{min} = 2$ and $n_{min} = 1$, as we need at least two rows to calculate a correlation. $\tau(m, n) = (n, n)$, because g takes a matrix of size $m \times n$ and returns a matrix of size $n \times n$.

6.3.2 Probabilistic model

Now, during exploration we can consider that not all primitives, and consistent sequences of primitives, are equally likely. Given our inputs: the hint text T, an input matrix A and partially filled output matrix B, we estimate the probability of a sequence of primitives, as follows:

$$p(g_1 g_2 \dots g_d | T, A, B) = \prod_{i=1}^{d} p(g_i | g_{i-1} g_{i-2} \dots g_1, T, A, B) \tag{6.1}$$

The expression on the right can be used partially as we include candidates for primitives during the search procedure.

In order to estimate this probability we consider the a priori probability $p(g)$ for each g, which we can derive from the frequency of use of the primitives in the library, as we will see in the following section. When T is available, we will use a frequency model that compares TF-IDF values (frequency of words appearing in the text) [152] of the primitive using the text from its R help documentation and the TF-IDF values from the text provided in T. This model produces the conditional probability $p_0(g|T)$ $\forall g, T$. We combine these probabilities as follows:

$$p(g|T, A, B) = \gamma p(g) + (1 - \gamma) p_0(g|T) \tag{6.2}$$

with $\gamma \in [0, 1]$. Clearly, if T is not available $\gamma = 1$. Basically, γ gauges how much relevance we give to the primitive prior (valid for all problems) over the relevance of the hint given by the user.

Finally, we have the intuition that the probability of a primitive may depend on the previous primitives. In this chapter, we explore a very simple model for sequential dependencies, by limiting the effect to trigrams and exploring whether the same primitive is repeated in any of the three previous operations. We use a parameter $\beta \in [0, 1]$, where high β values imply that repetitions are more penalised. More formally,

$$p(g_i | g_{i-1} \dots g_1, T, A, B) = \beta p(g|T, A, B) + (1 - \beta) p(g_i | g_{i-1} g_{i-2} g_{i-3}, T, A, B) \tag{6.3}$$

And the repetition part is simply:

$$\begin{aligned} p(g_i | g_{i-1} g_{i-2} g_{i-3}, T, A, B) &= 0 && \text{if } g_i \in \{g_{i-1}, g_{i-2}, g_{i-1}\} \\ &= p(g|T, A, B) && \text{otherwise} \end{aligned}$$

which means that if the primitive is repeated in the three previous operations, then the value is 0, becoming more relevant the lower β is. We will explore whether this repetition intuition has an important effect on the results.

6.3.3 Algorithm

With Eq. 6.1 using the expansion of Eq. 6.3, we can recalculate the probability after any primitive is introduced in a tree-based search. Note that every combination of primitives whose sizes do not match have probability 0 and are ruled out. However, for those that are valid, can we use extra heuristics to determine whether we are getting closer to the solution? One idea is to check whether we are approaching the final size of matrix. For instance, if the result has size $(m, 1)$ and an operation takes the dimension to exactly that, it may be more promising than another that leads to a size $(2m, n^3)$ (which would require further operations to be reduced, at least in size).

In particular, each node in the tree where functions $g_1 g_2 \dots g_d$ have been introduced will be assigned with the following priority[3]:

$$p^*(g_1 g_2 \dots g_d) = (1 + \alpha m) \prod_{i=1}^{d} p(g_i | g_{i-1} g_{i-2} \dots g_1, T, A, B) \tag{6.4}$$

Where $m = 1$ if the final dimension match the size of the output matrix B, i.e., $\tau_d(\tau_{d-1}(\dots \tau_1(m_{input}, n_{input}) \dots)) = (m_{output}, n_{output})$, and $m = 0$ otherwise. For those ongoing transformations where the output size matches (even if the values are not yet equal) the priority will be higher than if the dimensions do not match. In other words, it is just an estimate of whether "we may already be there". The parameter $\alpha \in [0, 1]$ simply gives weight to this. If $\alpha = 1$ then the priority of a situation with the final size is doubled over another situation where the final size does not match. For $\alpha = 0$ the priority is not affected by the final size matching or not.

Now we can use Eq. 6.4 in the tree-based search. The search algorithm works as follows (see Algorithm 1):

1. The system can be configured to use a set of primitive functions (G), for each of them including the minimum values for the size of the input (m_{min}, n_{min}) and the type function τ.

2. For each particular problem to solve, we take the input matrix A and the partially filled matrix B. Optionally, we take a text hint T describing the problem to solve.

3. Being d_{max} the maximum number of functions allowed in the solution, the procedure evaluates sequences of primitives $g_1 g_2 \dots g_d$, with $0 \leq d \leq d_{max}$

[3]This is an unnormalised value of the expectation that a partial primitive sequence could lead to the final solution, as used in a tree-based search. Because of the α correction, it may even be greater than 1, so it is not a probability.

where each $g_i \in G$. The parameter s_{max} determines the maximum number of solutions (when reached, the algorithm stops).

4. We start with a set of candidate solutions $C = G$.

5. We extract $c = g_1 g_2 \dots g_d \in C$ such that $p^*(c)$ is highest. We use τ on A and all primitives in c to see if the combination is feasible according to the dimension constraints and, in that case, we calculate the output size. If the dimension of any composition in c does not match, we delete the node from C. If the dimension of the output matches the dimension of B, we effectively execute the combination on A, i.e., $c(A)$, and check whether the result covers S, as defined in the previous section. In the positive case, we add c as a solution, and we delete it from C. In any other case, if $d < d_{max}$ we expand c into $c \cdot g_{d+1}$ with all $g_{d+1} \in G$. We calculate p^* for each of them and add them to C. We remove c from C.

6. We repeat the procedure in 5 above until s_{max} is reached or C is exhausted.

As mentioned in the problem formulation we allow for some small precision error ϵ between the cells in S (generated by $\hat{f}$) and the cells that are present in B (and are generated by $\hat{f}$).

6.3.4 Use of Text Hints

In some cases the user may provide a few words describing what she wants to do. This can be very helpful to give more relevance to those primitives that may be involved in the solution. For instance, if we consider a problem like "compute the correlation of a matrix", the primitive *cor* will probably appear in the solution. In our model, this is what we denoted $p_0(g|T)$. We now explain how we estimate this value.

First, we consider the set of primitives G and, for each of them, we download the text description from the corresponding R package *help* documentation. For instance `help("det")` gives the description for the function `det` as follows: *"det calculates the determinant of a matrix. determinant is a generic function that returns separately the modulus of the determinant, optionally on the logarithm scale, and the sign of the determinant"*. In the same way, the description of `diag` is: *"Extract or replace the diagonal of a matrix, or construct a diagonal matrix"*. Each of these help texts H_g is converted into an array by applying a bag-of-words transformation, after removing the useless words (included in a list of stop words) and performing a stemming conversion (reducing inflected words to their word stem).

Secondly, given a short description T of the task we want to solve, we also apply the bag-of-words transformation, remove the stop words and do stemming. Now we have the processed text chunks H_g for each $g \in G$ and the processed text chunk T. We extract the vocabulary V from all these text chunks.

Thirdly, we apply the TF-IDF conversion [152] to all vectors H_g and T using the same vocabulary V. TF-IDF gives more relevance to more informative words.

Algorithm 1
AUTOMAT[R]IX : Selecting matrix operations by example

Require: $A[m \times n]$ // *Input matrix*
Require: $B[m' \times n']$ // *Output matrix partially filled*
Require: G // *Primitives, defined as tuples* $\langle g, m_{min}, n_{min}, \tau \rangle$
Require: d_{max} // *Max primitives allowed in each solution*
Require: s_{max} // *Number of solutions allowed*
Ensure: Find a matrix $S \models_\epsilon B$
  $R \leftarrow \emptyset$ // *Initialise set of solutions*
  $C \leftarrow G$ // *Initialise candidate set C with list of functions*
  while $|R| \leq s_{max}$ `and` $C \neq \emptyset$ **do**
    $c \leftarrow argmax_{c \in C}\, p^*(c)$ // *Select the element c with the highest priority* p^*
    $d \leftarrow |c|$ // *Number of functions in c*
    $valid \leftarrow$ `True`
    $\langle m_{input}, n_{input} \rangle \leftarrow \tau(m, n)$
    for $i \leftarrow 1, d$ **do**
      $\langle g, m_{min}, n_{min}, \tau \rangle \leftarrow c[i]$ // *Extract the primitive and its type function*
      if $m_{input} < m_{min}$ `or` $n_{input} < n_{min}$ **then**
        $valid \leftarrow$ `False`
        $C \leftarrow C \backslash c$ // *Remove c from the set C*
        break
      end if
      $\langle m_{input}, n_{input} \rangle \leftarrow \tau(m_{input}, n_{input})$ // *Apply the type function* τ *to* m_{input}, n_{input}
    end for
    if $valid$ `and` $m_{input} = m'$ `and` $n_{input} = n'$ **then**
      $S \leftarrow apply(c, A)$ // *Run the sequence of functions in c over A*
      if $S \models_\epsilon B$ **then** // *Solution found*
        $R \leftarrow R \cup c$ // *Add c to the set of solutions*
      else
        if $d < d_{max}$ **then** // *Expand C*
          for $g \in G$ **do**
            $C \leftarrow c \circ g$ // *Create new combination and add it to C*
          end for
        end if
      end if
      $C \leftarrow C \backslash c$ // *Remove c from the set C*
    end if
  end while
  return R

This leads to a word vector h_g for each $g \in G$ and a word vector t. As an example, in Figure E.7 we can see the frequent terms for these two functions, as represented by their TF-IDF values. For instance, for the function `det`, it is clear that when the word *determinant* (and its steammed form *determin*) appears in a text hint, the function must have a higher probability of being required for the solution compared with other functions, such as the `diag` function.

Finally, for each g we calculate the cosine similarity $s(H_g, T)$ between h_g and t. We normalise the $|G|$ similarities to sum up to one as follows:

$$p_0(g|T) = \frac{s(H_g, T)}{\sum_{g \in G} s(H_g, T)} \tag{6.5}$$

This estimate is used for Eq. 6.2.

6.4 Experiments

We have implemented AUTOMAT[R]IX for R, a language and environment for statistical computing, data science and graphical representations. R operates on named data structures (vectors, matrices, data frames, etc.). In our case, we work with those functions such that input and output are data-related structures (matrices, vectors, etc.). These include primitives that extract characteristics of a matrix, such as number of rows or maximum value, or apply operations to the values, for instance bind columns, calculate the mean or compute a correlation matrix. More specifically, we take 34 R functions related to matrices from the *base*[4], *stats*[5] and *Matrix*[6] packages included in R. The experiments shown in this section aim to answer the following questions:

Q1 How much impact on success do the a priori probabilities have?

Q2 Are both α and β key to obtain better and faster results? How much?

Q3 Is the user text helpful and how much relevance (γ) should we give to it?

In order to answer these questions, in this section we describe how we obtain $p(g)$, the a priori distribution for the primitives, and how much impact has in the results; we describe the text models that lead to $p_0(g|T)$, we perform an

[4] https://stat.ethz.ch/R-manual/R-devel/library/base
[5] https://stat.ethz.ch/R-manual/R-devel/library/stats
[6] https://stat.ethz.ch/R-manual/R-devel/library/Matrix

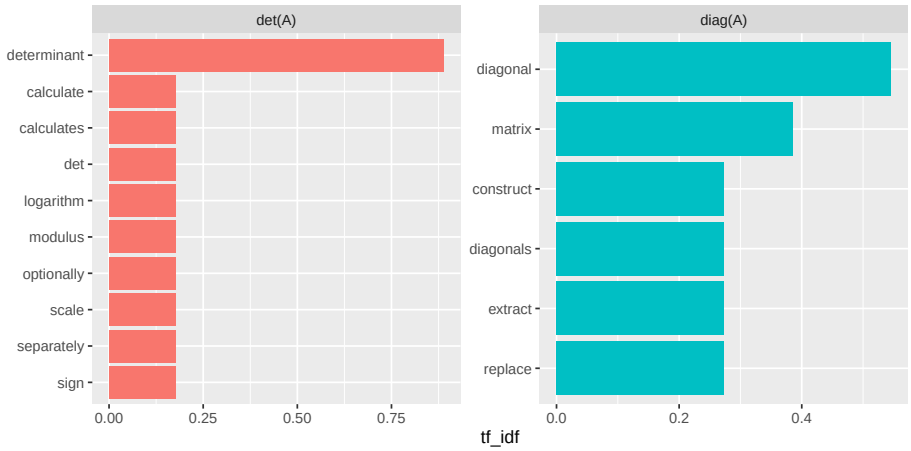


Figure 6.2. TF-IDF values for two R primitives extracted from the R help documentation.

ablation study to obtain the best values for parameters α, β and γ and then, we evaluate the algorithm with real problems taken from Stack Overflow[7].

6.4.0.1 A priori Probabilities: $p(g)$

To answer question **Q1** and calculate the a priori distribution we use the frequency of use of the functions to give different prior probabilities to each of them. To do this, we use the "Top 2000 most used R functions on GitHub" dataset, available for download on GitHub[8], containing an ordered list of the most frequently used functions by the programming community on Github. We reduce the 2000 functions to a subset containing only the functions included in our library G. When a function is duplicated in different packages we take one of them following this package order: *base*, *stats*, *Matrix*. Table 6.1 shows the six most used R functions from those functions included in G[9].

	function	p(g)
1	`length`	0.327975648
2	`nrow`	0.088785612
3	`is.na`	0.082437026
4	`max`	0.055495595
5	`mean`	0.046439995
6	`cbind`	0.045872954

Table 6.1. Top six R functions most used on GitHub.

Being n_g the absolute frequency of use for the function g, we calculate the a priori probability as follows:

$$p(g) = \frac{n_g}{\sum_{g \in G} n_g}$$

Benchmark: We will test whether the use of $p(g)$ as a very straightforward a priori probability is already useful in order to reduce the search time and space. For this, we first have tested the system with synthetic data. We have generated 10 random real matrices of different dimensions $m \times n$ where $m, n \in (2, 10)$. These matrices are filled with numeric values following a uniform distribution between 0 and 100. For each of these 10 matrices A we generate 10 transformations of depth $d = 1..4$ each using the functions from G according to their a priori probability (explained in section 6.4.0.2). Finally, for each of the $10 \times 10 \times 4 = 400$ matrices S we generate a matrix B where we replace a percentage uniformly chosen between 60% and 80% of the cells by empty values. In total we have 400 pairs of matrices A, B to test the algorithm with different numbers of operations.

[7] For replicability and to encourage future research, all the matrix transformations used here (the MATRANSF repository) and the code for AUTOMAT[R]IX are published on: https://github.com/liconoc/ProgramSynthesis-Matrix.

[8] Top 2000 R functions: shorturl.at/pDFRZ

[9] The complete list of the most used functions related to matrices can be found at Appendix E.2

Results: As we work with artificial examples with no text hints we cannot use $p_0(g|T)$ for these experiments. So, in this case we have tested the strategy using the a priori probabilities for $p(g)$ compared with a uniform baseline, assuming $p(g)$ uniform (so it is actually a breadth-first strategy). In both cases, we use $\gamma = 1$, $\alpha = \beta = 0$, $d_{max} = 4$, $s_{max} = 1$ (we only need to find one solution) and a timeout of 60 seconds[10].

strategy	accuracy	generated	explored	time
Uniform	0.47 ± 0.51	5610 ± 3543	168 ± 104	59.54 ± 46.78
Prior	**0.65 ± 0.49**	**790 ± 1266**	**24 ± 38**	**3.69 ± 7.01**

Table 6.2. Results for the synthetic examples. Average of generated and explored nodes, accuracy (percentage of correct solutions) and time in seconds, for the uniform and prior strategies. Experiments are performed with $d_{max} = 4$ and $s_{max} = 1$. The timeout is set to $60s$. Best results are highlighted in bold.

Table 6.2 shows the results in accuracy (percentage of correct solutions found, i.e., $\hat{f}(A) = S$), average of the number of nodes generated and actually explored, and time in seconds. From these results we can clearly see that the use of probabilities following the real use of the functions from GitHub has a great impact on the size of the space explored, reducing drastically the time needed to find one solution while improving the accuracy obtained. We state that the results using the *prior* strategy leads to significant results and, to prove it, we have run a significance test[69][11] The absolute value of the test statistic for our results, 1.022, is lower than the critical value of 1.694, so we reject the null hypothesis at the 0.05 significance level and accept the alternate mentioned, stating that the prior probability can help with obtaining better results by reducing the space of search.

6.4.0.2 Including Text Hints: $p_0(g|T)$

Benchmark: Now, we want to test the algorithm also including $p_0(g|T)$, i.e., using text hints provided by the user. To do this we need to find real matrix transformation problems. In this case, we have used questions and answers collected with the Stack Overflow API using the following parameters: `tagged="R"`, `title="matrix"` and `is_accepted_answer=1`. With these parameters we guarantee that (1) the posts received are answered and the answer is accepted by the creator of the post, (2) they are related to R and (3) they contain the term “Matrix" in the title. In total we collected 20 questions and answers of R problems dealing with matrix transformations. Just as example,

[10] Note that as B is partial, each problem can be consistent with many transformations, so the first solution may not be what the user expects (in our case the one we used to generate the example), i.e., $\hat{f}(A) \models_\epsilon B$, while $\hat{f}(A) \neq S$. In those cases more non-empty cells would be required to disambiguate the right solution, which may need more primitives.

[11] We have used the implementation of the Two Sample t-Test from the *stats* package from the R language.

some of the questions used are: *"How to reverse a matrix"*, *"Get positions for NA in a matrix"*, *"Rotate a Matrix in R"* or *"How to get the sum of each four rows of a matrix in R"*. Each example includes an input matrix A, an output matrix B, a text T (the title) and a solution. The solution is not used in the process, but just to generate B.

Computation Time and Parameter Settings: In order to answer questions **Q2** and **Q3**, we have performed an ablation study to determine the good ranges for the parameters α, β and γ. As said in the previous section, higher α increases the weight for those (partial) solutions that match the dimension of the output matrix, whereas higher β increases the penalisation for those solutions including repeated functions. Finally, higher values of γ give more weight to the prior probability over the text hint. In our ablation study, we consider all the possible permutations including values from 0 to 1 with increments of 0.25 each time.

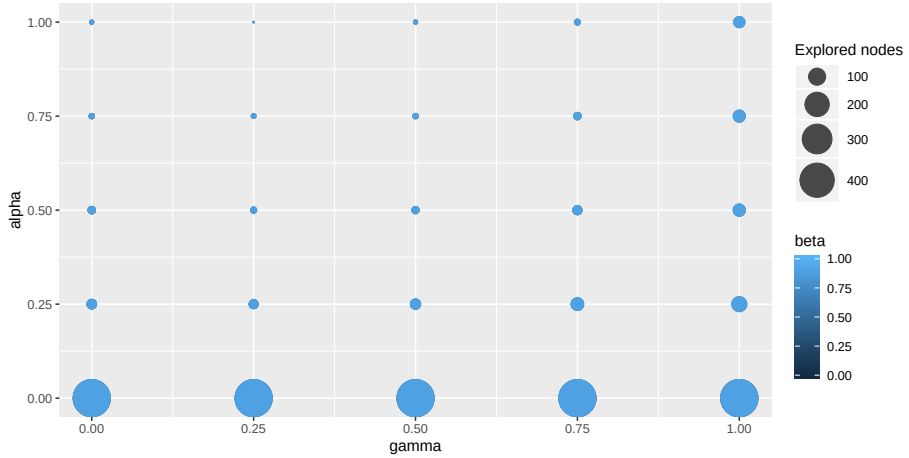


Figure 6.3. Average of the number of nodes that need to be explored to find the solution, depending on the value of α, β and γ parameters. Results using the MATRANSF data, $d_{max} = 4$ and $s_{max} = 1$.

Figure 6.3 shows the average number of explored nodes for the real problems when using different values for the parameters. Here, we can see that $alpha = 0$ increases drastically the nodes needed to find the right solution. In this case, the number of nodes is reduced considerably when $\alpha \geq 0.75$ (i.e., we give more relevance to the dimensions of the output matrix when building a partial solution during the search) and for values of $\gamma \leq 0.75$ (not giving all the relevance to the prior probabilty). We can see that, in this case, β seems to have no relevance in the number of nodes explored.

If fewer nodes have to be explored, the time needed can be reduced. Figure 6.4 shows the average of time needed for the experiments to find the solution depending on different values of α, β and γ. We can see clearly again that α is

very relevant to decrease the time needed to find the solution. The best times are obtained when $\gamma = 0.5$ and $\alpha = 1$. We can also see again that β seems to be not very useful in this study. However, although the β value on the figures seems completely flat, there are values that do differ, yet slightly, with times and number of nodes in ranges which are insignificant in comparison with the other two parameters. It seems that penalising (or not) the use of repeated primitives in a matrix pipeline does not have a visible effect on the results. The reason for this is that the examples extracted from StackOverflow do not have repeated functions in their solutions (since the solutions proposed in this forum tend to be short and efficient), in such a way that there is not a big difference in the results using β. In general, elegant solutions rarely have repeated primitives, except when these primitives are constants (e.g., "the element in the third row and the third column", using "3" in the solution as a repeated constant). However, we have not included constants in our list of functions because we would need to cherry pick a few small or frequent constants. This could be solved by a further study on how to include constants in the solutions without the use of explicit functions, by considering all constants in the algorithm itself but with an increasing cost for higher constants (lower probability). This would make the use and study of the beta parameter more insightful, with a controlled use of repetitions in the solutions. For now, we are assuming that β is no longer needed so we remove it from the experiments (i.e., $\beta = 0$).

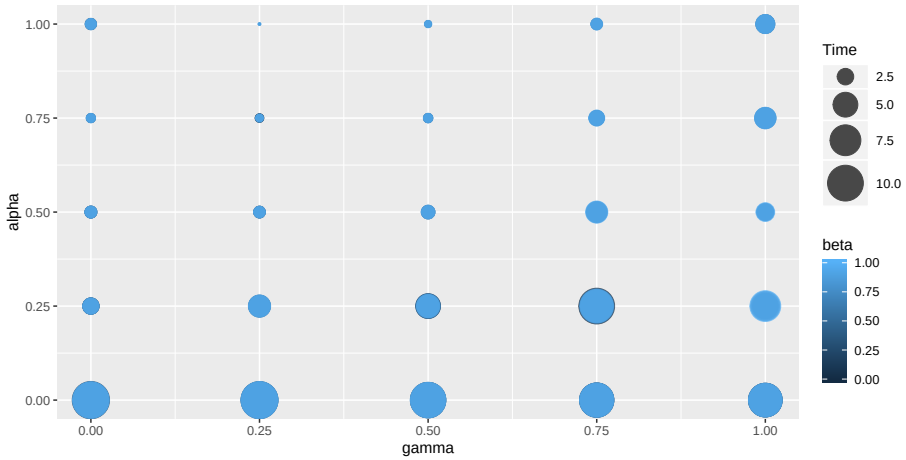


Figure 6.4. Average of the time needed to find the solution, depending on the value of α, β and γ parameters. Results using the MATRANSF data, $d_{max} = 4$ and $s_{max} = 1$.

With the ablation study, we see that most ranges of the parameters are safe, but some of the refinements we introduced are relevant (except the primitive repetition). We can decide the parameters and run the experiments using the best settings. Assuming that β is no longer needed we remove it from the experiments (i.e., $\beta = 0$). For the real-world datasets we have tested the strategy using

the a priori probabilities including the text hint and the calculated parameters (Prior+Text). We have compared this strategy with two baselines: the uniform and the a priori (without text hint) probabilities as they are explained in section 6.4.0.1. Concretely, we have tested the following strategies:

1. *Uniform (baseline)*: as said in section 6.4.0.1, we consider $p(g)$ uniform with $\gamma = 1$ and $\alpha = 0$.
2. *Prior (baseline)*: $p(g)$ is estimated with the a priori probabilities with $\gamma = 1$ and $\alpha = 0$.
3. *Prior+Text*: we use $p(g)$ as in the *Prior* strategy and $p(g|T)$ calculated using the text hint, with $\gamma = 0.5$ and $\alpha = 1$ (i.e., favouring solutions matching the dimension of B).

For each strategy we run the experiments with $d_{max} = 4$ and $s_{max} = 1$. We give a time-out of 120 seconds[12].

Results: We now analyse whether the new strategy is able to reduce the search space significantly. Table 6.3 shows the accuracy, nodes that generated and explored and running time for the new strategy compared with the uniform distribution (without hints or priors) and the a priori distribution. We can see that using a text hint, even when the hint can be really short (for instance, "Rotate a Matrix in R"), it helps to reduce the search space and the time needed, achieving higher accuracy. We can see these results better in Figure 6.5. Note that we are not introducing a new natural language for programming, we are just helping users to program certain matrix transformations simply by expressing what they need, as they might tell a human programmer. We need to keep in mind that in the case of R, the documentation of some functions is relatively short (and sometimes including more than one function per documentation) and therefore the description used contains very few words. It would be very useful to find larger texts that describe the functions in order to expand the range of words to better relate the text provided by the user with the functions of the background knowledge. Even so, we see that the search space is considerably reduced when the text hint is considered and α is higher. Using a uniform distribution the generated tree has (on average) almost 3500 nodes and in order to find the correct solution approximately 25% of the tree has to be explored. On the contrary, taking the text into account, the search tree is reduced to less than 500 nodes on average and we only need to explore 2% of this tree.

The obvious difference between these two strategies (*Uniform* and *Prior+Text*) derives from the fact that, with a uniform distribution, the search is performed depending on the order in which the functions have been included in the background knowledge. However, our strategy using the text no longer depends on the order of the background knowledge, since the search is reordered using

[12] A complete list of events produced during the execution of the algorithm for a simple example can be seen in Appendix E.5.

strategy	accuracy	generated	explored	time
Uniform	0.60 ± 0.50	3414 ± 3511	867 ± 549	57.91 ± 52.36
Prior	0.65 ± 0.49	1065 ± 201	620 ± 362	39.84 ± 43.73
Prior+Text	**1.00 ± 0.00**	**477 ± 429**	**14 ± 12**	**1.26 ± 1.18**

Table 6.3. Results for the examples from StackOverflow. Average and standard deviation for accuracy, number of generated and explored nodes, and time in seconds, broken down by the three strategies and $s_{max} = 1$. The timeout is set to 120s and $d_{max} = 4$. Best results are highlighted in bold.

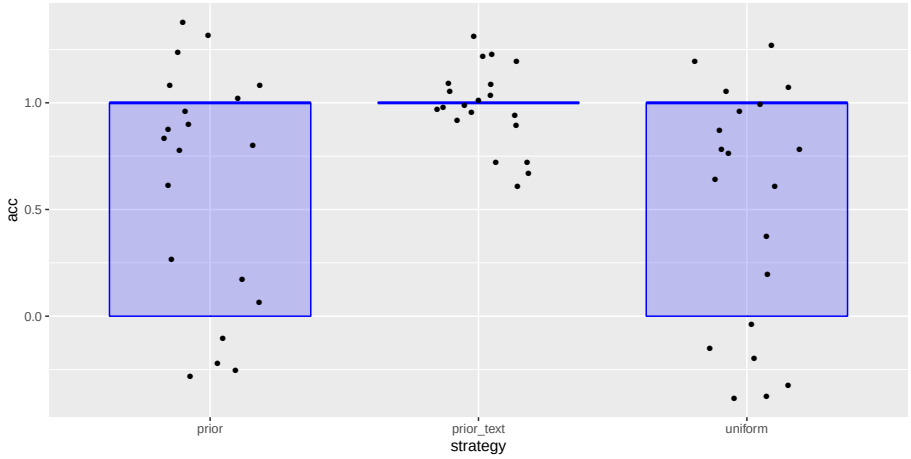


Figure 6.5. Average accuracy obtained using the different strategies.

the calculated probabilities, making the most probable solutions the first to be tested. Of course, the reduction of the search space has another consequence: a significant reduction in the computation time, causing that in some cases the problem is solved in less than a second. We can see these results better in Figure 6.6.

We can see the results in a different way in Figure 6.7. This plot shows on the y-axis the percentage of cases that are solved in less than the time expressed on the x-axis, which is the *timeout* (maximum time limit). Each of the treads shown in this Figure represents the average of time spent by each strategy to solve a different problem. Here, we can see clearly that the use of our strategy including priors and text hints, reduces the time needed to solve the problemsk, in such a way the system is able to solve all of them in less time that the time needed by the other strategies to solve the first of the problems.

Finally, Table 6.4 shows that the number of solutions over the pruned hypothesis given by the type function is a huge improvement over the initial 34^4 space. We can see that the average number of solutions found when running the experiments using $d_{max} = 4$ and $s_{max} = \infty$ (trying to find all the possible solutions), with a time-out of 120 seconds is 9.4, even when the average on the

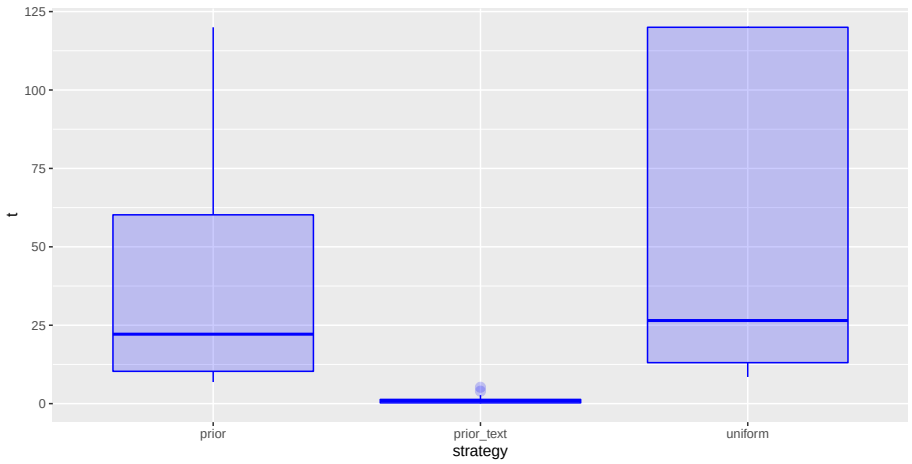


Figure 6.6. Average of time needed to solve the problems depending on the strategy used.

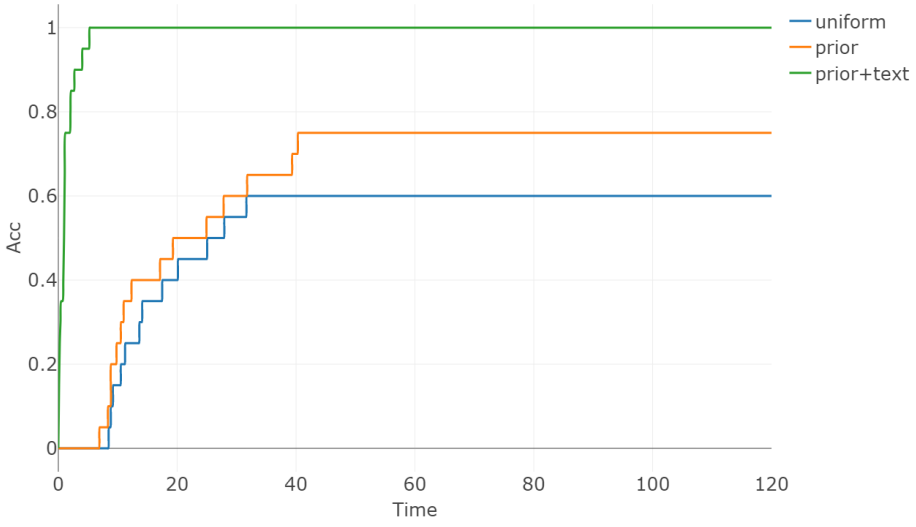


Figure 6.7. Percentage of cases (y-axis) that are solved in less than the time (seconds) on the x-axis depending on the strategy used. Results using $d_{max} = 4$ and $s_{max} = 1$.

number of explored nodes is 275. Note that with 34 primitives there are $34^{d_{max}}$ nodes to explore as a maximum. For example, with $d_{max} = 4$ this is 1,363,336 nodes. How can we get only an average of 867 explored with the uniform strategy, which is breadth-first? The answer is given by the combinations that are pruned because the type function does not match, and all the further reduction is given

by the use of probabilities in the other strategies. Here we can conclude that the pruning by the type function is the main reason for the reduction of the search space, and the reason is obvious: we are not testing the functions one by one until we find the correct one. First of all, we do not waste time actually trying all the combinations. Instead, the type function checks (without actually applying the functions) and based on the size of the input matrix and the size of the output matrix, whether, in case of applying the functions, the size of the resulting matrix would match the one we are looking for. Only in that case, the function (or combination of functions) will actually be applied to the input matrix to check if the result covers the output matrix.

example	n_{sols}	$n_{explored}$	$n_{created}$
1	1	267	9078
2	4	273	9180
3	16	285	9145
4	1	272	9282
5	2	273	9214
6	3	275	9235
7	3	268	8987
8	6	273	9050
9	1	270	9173
10	3	275	9224
11	2	268	9078
12	2	262	8874
13	2	265	8976
14	8	267	8806
15	32	292	8874
16	36	295	8806
17	28	286	8783
18	33	290	8747
19	2	272	9202
20	3	268	9078

Table 6.4. Number of solutions found for each example using $d_{max} = 4$ and $s_{max} = \infty$, with a $timeout = 120s$. n_{sols} represents the number of different solutions found; $n_{explored}$ is the total number of explored nodes; $n_{created}$ represents the total number of created nodes.

6.5 Conclusions

The process of generating code automatically can help data scientists when dealing with matrices (or data frames), the most common representation of information in data science, artificial intelligence and many other areas. In our vision, users would easily produce an example of the input matrix and a few cells of the output matrix, and the system will generate the code for them.

We have presented AUTOMAT[R]IX, a new system that is able to solve this very common problem of matrix transformation successfully. The system is based on a breadth-search approach pruned by the consistency of the types given by the dimensions of the matrices and the intermediate results. Besides, the system is guided by a strategy based on dynamic probabilities from a prior value depending on the frequency of the use of primitives on Github and, optionally, the relevance (TF-IDF) values of the terms in the text hints provided by the user compared with the terms in the R help documentation of the primitives.

We have tested the two baseline approaches with a synthetic dataset of 400 matrices and 34 different transformations in R. We have also compared several baselines compared with our algorithm using real examples of 20 problems extracted from Stack Overflow. The results show that the AUTOMAT[R]IX system is able to give the correct result for all of them in a very short time. In this case, using the strategy *Prior+Text* , which uses the text hints giving more importance to those solutions that match the dimension of the output matrix, AUTOMAT[R]IX can achieve 100% accuracy, reducing significantly the time spent to find the solution. Both datasets, with synthetic pipelines and real examples from Stack Overflow, are available as MATRANSF.

The type function used in our algorithm is clearly a perfect mechanism to reduce the time required to find the solution, while the use of probabilities based on the priors and the text provided by the user, is enormously useful to considerably reduce the search space. The combination of both strategies results in a system capable of finding the necessary functions to transform matrices in a fast and efficient way, reducing the size of the background knowledge, according to the restrictions of each of the functions included in it and the examples provided by the user.

Part V

Conclusions

Chapter 7

Conclusions and Future Work

In this chapter, we summarise the main contributions of this thesis, some lessons learnt and we identify some further work and recommendations.

7.1 Conclusions

7.1.1 Vision and starting point

Due to the high diversity of data sources, most data-related projects require the integration of data that are in different formats and/or in different data structures. This turns out to be a real problem as the current applications are not capable of dealing with raw data in inconsistent formats. Data scientists find themselves wasting their time cleaning, transforming and organising all these messy data. Data wrangling is the process inside data science that usually occupies more than the 80% of the time spent on the projects. *The solution to avoid this tremendous amount of manual work relies on automating data wrangling.*

We have seen that there are many data wrangling tasks. However, some of them can require more manual effort and/or domain knowledge than others. For instance, data introduced in free-text fields included in any survey or spreadsheet tend to be messy and include very different formats. For instance, this happens with dates in different standard or non-standard formats or with names having one or two surnames. Perhaps, the fastest solution to transform all of these data to a unified format relies on simply applying some functions in some programming language to the data, but here we need to assume that the user not only has knowledge about the domain of the data but also programming skills.

Inductive programming has been seen in recent years as an approach for data wrangling automation. Nevertheless, since the success of inductive programming strongly depends on having the right primitives in its background knowledge, one key issue of automating data wrangling is background knowledge. However, systems using inductive programming approaches suffer a real bottleneck when many primitives are included in the background knowledge, because they have too many of them to choose from. Automating (or semi-automating) data wrangling process requires to keep the size of the background knowledge at a very low level without affecting the effectiveness of the system for finding the right solution. *This thesis states that the answer for this problem relies on the creation of a dynamic background knowledge that can change depending on the problem to solve.*

7.1.2 Scientific insights and take-aways

As we saw in section 1.2, we set some objectives that could help us achieve the goal of successfully create a dynamic background knowledge. We have accomplished those objectives as follows:

- We have analysed the trajectories of data science projects and defined the role of data wrangling and its automation. We have seen that data wrangling not only happens at the beginning, but at many other points of the data science process.
- We have studied which data domains are normally used in data cleansing tools and what characteristics they have. *We have realised that problems belonging to the same domain tend to have similar syntactical characteristics. With this premise we have created the set of meta-features that are useful to differentiate one domain from the others.* We believe that this is a simple way to detect differences between domains, and the results are promising.
- We have collected or generated examples of common problems related to the transformation of data in different scenarios: (1) personal data; and (2) matrix transformations. Through the collection of the data *we have realised that, although most projects require a lot of time to fix the data before use, data scientists do not document the process they go through to fix these data. Moreover, only the clean datasets are published.*
- We have generated a domain-specific background knowledge for each domain, including a large set of functions in each of them. We have organised them into predefined domains the user can create, and modified, and we have also explored the option of using rankings of primitives.
- *We have generated algorithms to dynamically handle the background knowledge. We have tested several strategies to select or construct the appropriate background knowledge based on the characteristics of the data wrangling problems.* The use of meta-features, as well as the use of tips in natural language, are strategies that aim to reduce the size of the background knowledge for any particular problem.
- *We have provided two systems based on inductive programming, which use the algorithms and the generated background knowledge, in order to solve the collected problems, using as input as few examples as possible.*

The algorithms proposed in this thesis considerably reduce the background knowledge dynamically, allowing us to solve most of the problems we have tested. *We have shown that data wrangling automation is possible through a dynamically-selected background knowledge.* We believe it can be helpful for the reduction of the inductive bias in any kind of area.

We have shown that *fully automated systems for data wrangling could be possible if the background knowledge has enough information about the domain and the context of the problems.* However, when this is not possible, *we have learnt that user action is required in the form of prior information before the process.* In this thesis we have seen that a few input/output examples or tips are

enough for a semi-automatic system. Likewise, at the beginning we considered that the expert's feedback (after the transformations) would also be important, since in problems related to data wrangling, the domain expert usually performs fine-grained checks at the end. However, after our experiments *we have also learnt that feedback is not always required to obtain the best results, but it can be a useful help when a problem cannot be fully automated or the system is not able to find a right solution* (the detected domain is not the correct one).

Based on our experiments, *the work can be replicated and expanded to other areas* where programming in R (or other languages) is required, or even to help people who are starting to learn how to program from scratch (for instance, on fields like bioinformatics, where biologists, doctors, chemists, and other specialists try to start working with R or python).

Beyond the application to data wrangling, *we see that the effective combination of background knowledge and hypothesis-driven*[1] *learning is a particularly promising niche where other areas inside or outside AI, or machine learning alone, are having more difficulties, especially when only few examples are available.* For instance, learning how to generate new handwritten characters given a few examples, by combining small parts or pieces of them. We hope the ideas presented in this thesis could be useful and could be applied for many other areas and extended by further researches.

7.1.3 Scientific Contributions

Overall, this thesis contains several contributions:

- **Identification of most common data domains in data wrangling:** we study the current tools for data wrangling as well as help forums to detect which are the most usual problems, domains and data formats when dealing with data wrangling problems.
- **New data wrangling dataset repository:** we provide a set of datasets as an extended open benchmark specifically designed for the evaluation of data wrangling tools focusing on column transformation problems, for further progress, replicability and comparison in this area. We also provide a collection of matrix pipeline transformations, MATRANSF, some of them with textual hints, organised as a benchmark repository for the community.
- **Better understanding of the trade-off between breadth and depth as inductive bias:** we analyse how the breadth, depth and number of instances affect the efficiency, showing that we can achieve a trade-off between breadth and depth, and still solve many problems using only one example.
- **New strategies for dynamically handling background knowledge:** We propose several strategies to *dynamically* select or construct the

[1] Generating new data for an existing hypothesis.

appropriate background knowledge automatically following the idea of detecting the best specialised functions according to the context of the particular problem to solve. First, we adapt the search space according to the domain of the data, generating different DSBKs. Second, we reduce the background knowledge by a ranking of functions based on meta-features of the examples. Third, we also include the combination of both approaches to reduce the search space only to the most probable functions included in the DSBK. Finally, we present an algorithm able to order and reduce the search space of primitives by using constraints of the problems, some prior probabilities and text hints provided by the user in natural language.

- **Two new data wrangling systems, BK-ADAPT and AUTOMAT[R]IX:** BK-ADAPTis a system to guide the search through the background knowledge by extracting several meta-features from the examples. AUTOMAT[R]IXis a learning algorithm guided by a tree-based search that uses the dimensions of the examples, prior primitive probabilities and textual hints to learn the transformations needed efficiently. With BK-ADAPT and AUTOMAT[R]IX, we show that these generic and semi-automatic approaches, combining inductive programming with appropriate operators to define and reduce the necessary background knowledge, are able to improve the results of other state-of-the-art —and more specific— data wrangling approaches.

7.2 Future Work

There are still many questions and problems that need to be addressed, but we hope that, for future researches, this manuscript proposes new directions for task automation. By now, we can see several interesting future lines of research. Let us enumerate some of them:

- **Improve understanding of domains**: the domain detection mechanism should be improved. One way to better address this problem would be to explore ways to automatically learn about the characteristics of the domain or context of the data to work with. In the case that there is no knowledge about that domain, we would need to be able to learn automatically from it to generate new knowledge that can be used in future problems of the same domain or similar domains. For example, it would be very useful for the machine to be able to incrementally learn each of these domains, that is, to be able to read examples of data, recognise their main characteristics (syntactic and semantic) and classify and store them by "similar domains" (as Google Photos[2] face detection does with new photos). With this classification, a grammar could be created and dynamically updated and re-organised in an incremental way as new data arrived.

[2]Google Photos: https://photos.google.com/

- **Data scientist's competences, skills and actions:** we have seen that, in the case of data wrangling, there is limited documentation or public data to better understand the process. In this sense, two ways to help improve and expand the existing knowledge about the data wrangling process would be: (1) study the skills or abilities that are necessary to carry out the process (presented in Appendix B), so that we can investigate in what way it is possible to automate them or make them simpler; and (2) share or monitor the actions that users who work with data take when they have to deal with problems related to data wrangling (presented in Appendix C). Collecting and studying these two types of information could be very useful in future research to automate the data science process, and data wrangling in particular.

- **BK-ADAPT:** in the particular case of the first system created for this thesis, it could be useful to consider other ways of solving the ranking of functions to avoid a fixed value of the maximum functions to combine. For instance, using a dynamic threshold based on the probabilities returned by the *primitive estimator*. Besides, it could be useful to have more than one input to generate the output (for instance a user name and a domain as inputs and an email as output). Of course, we have only seen a small number of domains and many more should be studied and incorporated in BK-ADAPT to make it useful for many more people.

- **AUTOMAT[R]IX:** in the case of AUTOMAT[R]IX, we consider to add new constraints over the data dimensionality, or over the values (positive values only). We can also explore more efficient algorithms in such a way we can add constants (arguments for the functions) or multiple pairs of input-output matrices. Of course, the approach can be replicated to synthesise functions with other languages such as Python. Besides, generating an R package for AUTOMAT[R]IX would be tremendously useful to help people who get stuck because they do not know how to program.

- **Application in other areas:** we would like to extend the use of the algorithms and the research done for this thesis, and apply them to other problems from different fields. Concretely, we are primarily interested in some areas of bioinformatics, such as functional genomics (field that tries to understand how the molecular components of living organisms interact together to form living organisms), where inductive programming and a dynamic background knowledge could be useful.

As a final observation, in this thesis we have seen that the automation of the data science process, and data wrangling in particular, has been improved in recent years but it is still a very immature field that has a long way to go. We hope the approaches, findings and benchmarks generated for this thesis will be useful for the future research in the area.

Bibliography

[1] *8 Skills You Need to Be a Data Scientist*. en-US. Nov. 2014. URL: https://blog.udacity.com/2014/11/data-science-job-skills.html.

[2] Wil van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016.

[3] David H Autor and David Dorn. "The growth of low-skill service jobs and the polarization of the US labor market". In: *American Economic Review* 103.5 (2013), pp. 1553–97.

[4] Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. "Deepcoder: Learning to write programs". In: *arXiv preprint arXiv:1611.01989* (2016).

[5] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. "FlashRelate: extracting relational data from semi-structured spreadsheets using examples". In: *ACM SIGPLAN Notices*. Vol. 50. 6. ACM. 2015, pp. 218–228.

[6] Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. "Deep API Programmer: Learning to Program with APIs". In: *arXiv preprint arXiv:1704.04327* (2017).

[7] Alan W Biermann. "The inference of regular LISP programs from examples". In: *IEEE transactions on Systems, Man, and Cybernetics* 8.8 (1978), pp. 585–600.

[8] Alan W Biermann, Gérard Guiho, and Yves Kodratoff. *Automatic program construction techniques*. Macmillan New York, 1984.

[9] Svetla Boytcheva. "Overview of ILP Systems". In: *Journal of Cybernetics and Information Technologies* (2002).

[10] Martin Braschler, Thilo Stadelmann, and Kurt Stockinger. *Applied Data Science*. Springer, 2019.

[11] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[12] Afroz Chakure. *Random Forest Regression: Along with its implementation in Python*. 2019.

[13] Wo L Chang, Nancy Grady, et al. *NIST Big Data Interoperability Framework: Volume 1, Big Data Definitions*. Tech. rep. 2015.

[14] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. "CRISP-DM 1.0 Step-by-step data mining guide". In: (2000).

[15] Roger Chua. *Supervised Learning*. May 2019. URL: https://becominghuman.ai/a-simple-way-to-explain-how-machines-learn-7d9155ac8bed.

[16] Comisión Europea and Dirección General de Educación y Cultura. *The european qualifications framework for lifelong learning (EFQ)*. en. OCLC: 630826756. Luxembourg: Office for Official Publications of the European Communities, 2008. ISBN: 978-92-79-08474-4.

[17] Lidia Contreras-Ochando. "Automated Data Transformation with Inductive Programming and Dynamic Background Knowledge". In: *Approaches and Applications of Inductive Programming (Dagstuhl Seminar 19202)*. Ed. by Luc De Raedt, Richard Evans, Stephen H. Muggleton, and Ute Schmid. Vol. 9. 5. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 70–71. DOI: 10.4230/DagRep.9.5.58. URL: http://drops.dagstuhl.de/opus/volltexte/2019/11381.

[18] Lidia Contreras-Ochando. "Domain Specific Induction for Data Wrangling Automation". In: *Approaches and Applications of Inductive Programming (Dagstuhl Seminar 17382)*. Ed. by Ute Schmid, Stephen H. Muggleton, and Rishabh Singh. Vol. 7. 9. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 94–95. DOI: 10.4230/DagRep.7.9.86. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8590.

[19] Lidia Contreras-Ochando, César Ferri, and José Hernández-Orallo. "AUTOMAT[R]IX: Learning Simple Matrix Pipelines". In: Machine Learning (to appear). 2020.

[20] Lidia Contreras-Ochando, Cèsar Ferri, and José Hernández-Orallo. "Automating Common Data Science Matrix Transformations". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2019, pp. 17–27.

[21] Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, and Susumu Katayama. "Automated Data Transformation with Inductive Programming and Dynamic Background Knowledge". In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, ECML PKDD 2019*. ECML-PKDD '19. Germany, 2019.

[22] Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, and Susumu Katayama. "BK-ADAPT: Dynamic Background Knowledge for Automating Data Transformation". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2019, pp. 755–759.

[23] Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, and Susumu Katayama. "Domain specific induction for data wrangling automation". In: *AutoML@ ICML, Sydney, Australia* (2017).

[24] Lidia Contreras-Ochando, César Ferri, José Hernández-Orallo, Fernando Martínez-Plumed, María José Ramírez-Quintana, and Susumu Katayama. "General-purpose Declarative Inductive Programming with Domain-Specific Background Knowledge for Data Wrangling Automation". In: *arXiv preprint arXiv:1809.10054* (2018).

[25] Lidia Contreras-Ochando, Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, and María José Ramírez-Quintana. "General-Purpose Inductive Programming for Data Wrangling Automation". In: *AI4DataSci @ NIPS 2016* (2016).

[26] Lidia Contreras-Ochando, Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, and María José Ramírez-Quintana. "Logging Data Scientists: Collecting Evidence for Data Science Automation". In: *AI4DataSci @ NIPS 2016* (2016).

[27] Claire D. Costa. *Top Programming Languages for Data Science in 2020*. 2020.

[28] Andrew Cropper. "Forgetting to learn logic programs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3676–3683.

[29] Andrew Cropper, Alireza Tamaddoni, and Stephen H Muggleton. "Meta-interpretive learning of data transformation programs". In: *Inductive Logic Programming*. 2015, pp. 46–59.

[30] Tamraparni Dasu and Theodore Johnson. *Exploratory data mining and data cleaning*. Vol. 479. John Wiley & Sons, 2003.

[31] *Data types in Data Models - Microsoft Excel Documentation*. en-US. URL: https://support.office.com/en-us/article/data-types-in-data-models-e2388f62-6122-4e2b-bcad-053e3da9ba90.

[32] Tijl De Bie, Luc De Raedt, Holger H. Hoos, and Padhraic Smyth. "Automating Data Science (Dagstuhl Seminar 18401)". In: *Dagstuhl Reports* 8.9 (2019). Ed. by Tijl De Bie, Luc De Raedt, Holger H. Hoos, and Padhraic Smyth, pp. 154–181. ISSN: 2192-5283. DOI: 10.4230/DagRep.8.9.154. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10344.

[33] Eyal Dechter, Jonathan Malmaud, Ryan Prescott Adams, and Joshua B Tenenbaum. "Bootstrap learning via modular concept discovery". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. AAAI Press/International Joint Conferences on Artificial Intelligence. 2013.

[34] Yuri Demchenko, Adam Belloum, and Tomasz Wiktorski. "The second machine age". In: *Education for Data Intensive Science to Open New science frontiers* (2017).

[35] Li Deng and Dong Yu. *Deep Learning: Methods and Applications*. Tech. rep. MSR-TR-2014-21. Microsoft, May 2014. URL: https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/.

[36] Jacob Devlin, Rudy R Bunel, Rishabh Singh, Matthew Hausknecht, and Pushmeet Kohli. "Neural Program Meta-Induction". In: *NIPS*. 2017, pp. 2077–2085.

[37] Vasant Dhar. "Data science and prediction". In: *Communications of the ACM* 56.12 (2013), pp. 64–73.

[38] Sebastijan Dumancic and Andrew Cropper. *Knowledge Refactoring for Program Induction*. 2020. arXiv: 2004.09931 [cs.AI].

[39] Organisation for Economic Co-operation and Development (OECD). *Automation and independent work in a digital economy: policy brief on the future of work*. 2016. URL: https://www.oecd.org/els/emp/Policy%5C%20brief%5C%20-%5C%20Automation%5C%20and%5C%20Independent%5C%20Work%5C%20in%5C%20a%5C%20Digital%5C%20Economy.pdf.

[40] Organisation for Economic Co-operation and Development (OECD). *Putting faces to the jobs at risk of automation*. 2018. URL: https://www.oecd.org/employment/Automation-policy-brief-2018.pdf.

[41] Kevin Ellis and Sumit Gulwani. "Learning to learn programs from examples: Going beyond program structure". In: *International Joint Conference on Artifical Intelligence (IJCAI)*. 2017.

[42] Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. "Learning libraries of subroutines for neurally–guided bayesian program induction". In: *Advances in Neural Information Processing Systems*. 2018, pp. 7805–7815.

[43] *European e-Competence Framework*. en-US. URL: http://www.ecompetences.eu/.

[44] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "From data mining to knowledge discovery in databases". In: *AI magazine* 17.3 (1996), p. 37.

[45] César Ferri-Ramírez, José Hernández-Orallo, and María José Ramírez-Quintana. "Incremental learning of functional logic programs". In: *International Symposium on Functional and Logic Programming*. Springer. 2001, pp. 233–247.

[46] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[47] Pierre Flener. "Achievements and prospects of program synthesis". In: *Computational logic: logic programming and beyond*. Springer, 2002, pp. 310–346.

[48] Pierre Flener. “Inductive logic program synthesis with DIALOGS”. In: *International Conference on Inductive Logic Programming*. Springer. 1996, pp. 175–198.

[49] Pierre Flener and Ute Schmid. “An introduction to inductive programming”. In: *Artificial Intelligence Review* 29.1 (2008), pp. 45–62.

[50] Pierre Flener and Serap Yíilmaz. “Inductive synthesis of recursive logic programs: Achievements and prospects”. In: *The Journal of Logic Programming* 41.2-3 (1999), pp. 141–195.

[51] Jake Frankenfield. *Artificial Intelligence (AI)*. 2020.

[52] Carl Benedikt Frey and Michael A Osborne. “The future of employment: How susceptible are jobs to computerisation?” In: *Technological forecasting and social change* 114 (2017), pp. 254–280.

[53] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. “AJAX: An Extensible Data Cleaning Tool”. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’00. Dallas, Texas, USA: ACM, 2000, pp. 590–. ISBN: 1-58113-217-4. DOI: 10.1145/342009.336568. URL: http://doi.acm.org/10.1145/342009.336568.

[54] Alison Gopnik. “Making AI more human”. In: *Scientific American* 316.6 (2017), pp. 60–65.

[55] Georg Gottlob, Nicola Leone, and Francesco Scarcello. “On the complexity of some inductive logic programming problems”. In: *International Conference on Inductive Logic Programming*. Springer. 1997, pp. 17–32.

[56] Laura A Granka, Thorsten Joachims, and Geri Gay. “Eye-tracking analysis of user behavior in WWW search”. In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2004, pp. 478–479.

[57] David Guest. “The hunt is on for the Renaissance Man of computing”. In: *The Independent* 17.09 (1991).

[58] Sumit Gulwani. “Applications of Inductive Programming in Data Wrangling”. In: *Talks at Dagstuhl seminar on Approaches and Applications of Inductive Programming*. Oct. 2015.

[59] Sumit 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. Austin, Texas, USA: ACM, 2011, pp. 317–330.

[60] Sumit Gulwani. “Programming by Examples”. In: *Dependable Software Systems Engineering* 45 (2016), p. 137.

[61] Sumit Gulwani, William R Harris, and Rishabh Singh. “Spreadsheet data manipulation using examples”. In: *Communications of the ACM* 55.8 (2012), pp. 97–105.

[62] Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen Muggleton, Ute Schmid, and Benjamin Zorn. "Inductive programming meets the real world". In: *Communications of the ACM* 58.11 (2015), pp. 90–99.

[63] Philip J Guo, Sean Kandel, Joseph M Hellerstein, and Jeffrey Heer. "Proactive wrangling: mixed-initiative end-user programming of data transformation scripts". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM. 2011, pp. 65–74.

[64] Isabelle Guyon, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Núria Macià, Bisakha Ray, Lukasz Romaszko, Michèle Sebag, et al. "A Brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning Without Human Intervention". In: *Workshop on Automatic Machine Learning*. 2016, pp. 21–30.

[65] Kelli Ham. "OpenRefine (version 2.5). http://openrefine. org." In: *Journal of the Medical Library Association: JMLA* 101.3 (2013), p. 233.

[66] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[67] Xin He, Kaiyong Zhao, and Xiaowen Chu. *AutoML: A Survey of the State-of-the-Art*. 2020. arXiv: 1908.00709 [cs.LG].

[68] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. "Transform-data-by-example (TDE): an extensible search engine for data transformations". In: *Proceedings of the VLDB Endowment* 11.10 (2018), pp. 1165–1177.

[69] Nathanael A. Heckert, James J. Filliben, C M. Croarkin, B Hembree, William F. Guthrie, P Tobias, and J Prinz. *NIST/SEMATECH e-Handbook of Statistical Methods*. 2012.

[70] Jeffrey Heer, Joseph M. Hellerstein, and Sean Kandel. "Data Wrangling". In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Zomaya. Cham: Springer International Publishing, 2018, pp. 1–8. ISBN: 978-3-319-63962-8. DOI: 10.1007/978-3-319-63962-8_9-1. URL: https://doi.org/10.1007/978-3-319-63962-8_9-1.

[71] Robert Henderson. "Incremental learning in inductive programming". In: *International Workshop on Approaches and Applications of Inductive Programming*. Springer. 2009, pp. 74–92.

[72] José Hernandez-Orallo and María José Ramirez-Quintana. "Inverse narrowing for the inductive inference of functional logic programs". In: *Proceedings of the 1998 Joint Conference of Declarative Programming, APPIA-GULP-PRODE*. Vol. 98. Citeseer. 2011.

[73] José Hernández-Orallo. "Deep knowledge: Inductive programming as an answer". In: *Approaches and Applications of Inductive Programming (Dagstuhl Seminar 13502)*. Ed. by Sumit Gulwani, Emanuel Kitzelmann, and Ute Schmid. Vol. 3. 12. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 43–66. DOI: 10.4230/DagRep.3.12.43. URL: http://drops.dagstuhl.de/opus/volltexte/2014/4507.

[74] Jose Hernández-Orallo. *Measuring Intelligence Incrementally: Search, Demonstration and Transmission*. 2020.

[75] José Hernández-Orallo, Stephen H Muggleton, Ute Schmid, and Benjamin Zorn. "Approaches and applications of inductive programming (Dagstuhl seminar 15442)". In: *Dagstuhl Reports*. Vol. 5. 10. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.

[76] José Hernández-Orallo and María José Ramírez-Quintana. "A strong complete schema for inductive functional logic programming". In: *International Conference on Inductive Logic Programming*. Springer. 1999, pp. 116–127.

[77] José Hernández-Orallo and María José Ramírez-Quintana. "Inverse Narrowing for the Induction of Functional Logic Programs." In: *APPIA-GULP-PRODE*. Citeseer. 1998, pp. 379–392.

[78] Alan Hevner and Samir Chatterjee. "Design science research in information systems". In: *Design research in information systems*. Springer, 2010, pp. 9–22.

[79] Martin Hilbert and Priscila López. "The world's technological capacity to store, communicate, and compute information". In: *science* 332.6025 (2011), pp. 60–65.

[80] Martin Hofmann, Emanuel Kitzelmann, and Ute Schmid. "A unifying framework for analysis and evaluation of inductive programming systems". In: *Proceedings of the 2nd Conference on Artificiel General Intelligence (2009)*. Atlantis Press. 2009.

[81] Gretchen Huizinga and Sumit Gulwani. *Program synthesis and the art of programming by intent with Dr. Sumit Gulwani*. Nov. 2019. URL: https://www.microsoft.com/en-us/research/podcast/program-synthesis-and-the-art-of-programming-by-intent-with-dr-sumit-gulwani/.

[82] Earl B Hunt, Janet Marin, and Philip J Stone. *Experiments in induction*. Academic press, 1966.

[83] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[84] Laura Igual and Santi Seguí. "Introduction to Data Science". In: *Introduction to Data Science*. Springer, 2017, pp. 1–4.

[85] Tony Jenkins. "On the difficulty of learning to program". In: *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. Vol. 4. 2002. Citeseer. 2002, pp. 53–58.

[86] Cecily Josten and Grace Lordan. “Robots at Work: Automatable and non-automatable Jobs”. In: *Handbook of Labor, Human Resources and Population Economics* (2020), pp. 1–24.

[87] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono. “Research directions in data wrangling: Visualizations and transformations for usable and credible data”. In: *Inf. Visualization* 10.4 (2011), pp. 271–288.

[88] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. “Wrangler: Interactive visual specification of data transformation scripts”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 3363–3372.

[89] Gordon V Kass. “An exploratory technique for investigating large quantities of categorical data”. In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29.2 (1980), pp. 119–127.

[90] Susumu 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*. 2012, pp. 43–52.

[91] Susumu Katayama. “Systematic search for lambda expressions”. In: *In Proceedings Sixth Symposium on Trends in Functional Programming (TFP2005*. 2005.

[92] Susumu Katayama. “Systematic search for lambda expressions.” In: *Trends in functional programming* 6 (2005), pp. 111–126.

[93] *Key Features of RapidMiner Studio*. URL: https://rapidminer.com/products/studio/feature-list/.

[94] Jun H Kim, Daniel V Gunn, Eric Schuh, Bruce Phillips, Randy J Pagulayan, and Dennis Wixon. “Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems”. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM. 2008, pp. 443–452.

[95] Emanuel Kitzelmann. “Data-driven induction of recursive functions from input/output-examples”. In: *Proceedings of the ECML/PKDD 2007 Workshop on Approaches and Applications of Inductive Programming (AAIP’07)*. 2007, pp. 15–26.

[96] Emanuel Kitzelmann. “Inductive programming: A survey of program synthesis techniques”. In: *International workshop on approaches and applications of inductive programming*. Springer. 2009, pp. 50–73.

[97] Emanuel Kitzelmann and Ute Schmid. “Inductive synthesis of functional programs: An explanation based generalization approach”. In: *Journal of Machine Learning Research* 7.Feb (2006), pp. 429–454.

[98] Martin Koehler, Edward Abel, Alex Bogatu, Cristina Civili, Lacramioara Mazilu, Nikolaos Konstantinou, Alvaro Fernandes, John Keane, Leonid Libkin, and Norman W Paton. "Incorporating Data Context to Cost-Effectively Automate End-to-End Data Wrangling". In: *IEEE Computer Architecture Letters* 01 (2019), pp. 1–1.

[99] Janez Kranjc, Vid Podpečan, and Nada Lavrač. "Clowdflows: a cloud based scientific workflow platform". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2012, pp. 816–819.

[100] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338.

[101] Vu Le and Sumit 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. Edinburgh, United Kingdom: ACM, 2014, pp. 542–553. ISBN: 978-1-4503-2784-8. DOI: 10.1145/2594291.2594333. URL: http://doi.acm.org/10.1145/2594291.2594333.

[102] Yann LeCun. *Facebook's 'Deep Learning' Guru Reveals the Future of AI*. Dec. 2013. URL: https://www.wired.com/2013/12/facebook-yann-lecun-qa/.

[103] Jorge Leonel. *A simple way to explain how machines learn in the AI world*. June 2018. URL: https://becominghuman.ai/a-simple-way-to-explain-how-machines-learn-7d9155ac8bed.

[104] Henry Lieberman. *Your wish is my command: Programming by example*. Morgan Kaufmann, 2001.

[105] James Robert Lloyd, David K Duvenaud, Roger B Grosse, Joshua B Tenenbaum, and Zoubin Ghahramani. "Automatic Construction and Natural-Language Description of Nonparametric Regression Models." In: *AAAI*. 2014, pp. 1242–1250.

[106] John Wylie Lloyd. *Foundations of Logic Programming*. 2nd. Springer-Verlag New York, Inc., 1993. ISBN: 0387181997.

[107] Steve Lohr. "For big-data scientists,'janitor work'is key hurdle to insights". In: *New York Times* 17 (2014), B4.

[108] Mike Loukides. *What Is Data Science?* en. "O'Reilly Media, Inc.", Apr. 2011. ISBN: 978-1-4493-3609-7.

[109] Michael R Lowry and Robert D McCartney. "Automating software design". In: American Association for Artificial Intelligence. 1991.

[110] Zohar Manna and Richard Waldinger. *Fundamentals of deductive program synthesis*. Tech. rep. Stanford University CA, Department of Computer Science, 1992.

[111] Oscar Marbán, Javier Segovia, Ernestina Menasalvas, and Covadonga Fernández-Baizán. "Toward data mining engineering: A software engineering approach". In: *Information systems* 34.1 (2009), pp. 87–107.

[112] Fernando Martínez-Plumed, Lidia Contreras Ochando, César Ferri, Peter A. Flach, José Hernández-Orallo, Meelis Kull, Nicolas Lachiche, and María José Ramírez-Quintana. "CASP-DM: Context Aware Standard Process for Data Mining". In: *CoRR* abs/1709.09003 (2017). arXiv: 1709.09003. URL: http://arxiv.org/abs/1709.09003.

[113] Henar Martín, Ana M Bernardos, Josué Iglesias, and José R Casar. "Activity logging using lightweight classification techniques in mobile devices". In: *Personal and ubiquitous computing* 17.4 (2013), pp. 675–695.

[114] Fernando Martínez-Plumed, Lidia Contreras-Ochando, César Ferri, José Hernández Orallo, Meelis Kull, Nicolas Lachiche, Maréa José Ramírez Quintana, and Peter A Flach. "CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories". In: *IEEE Transactions on Knowledge and Data Engineering* (2019).

[115] Fernando Martínez-Plumed, César Ferri, José Hernández-Orallo, and María José Ramírez-Quintana. "Knowledge acquisition with forgetting: an incremental and developmental setting". In: *Adaptive Behavior* 23.5 (2015), pp. 283–299.

[116] Fernando Martínez-Plumed, César Ferri, José Hernández-Orallo, and María José Ramírez-Quintana. "Learning with configurable operators and rl-based heuristics". In: *International Workshop on New Frontiers in Mining Complex Patterns*. Springer. 2012, pp. 1–16.

[117] Andrew McAfee and Erik Brynjolfsson. "EDISON Data Science Framework: Part 1. Data Science Competence Framework (CF-DS) Release 2". In: *Wie die nächste digitale* (2014).

[118] Luigi Federico Menabrea and Ada Lovelace. *Sketch of the analytical engine invented by Charles Babbage*. 1842.

[119] Aditya Menon, Omer Tamuz, Sumit Gulwani, Butler Lampson, and Adam Kalai. "A machine learning framework for programming by example". In: *ICML*. 2013, pp. 187–195.

[120] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bo Yang, Justin Betteridge, Andrew Carlson, B Dalvi, Matt Gardner, Bryan Kisiel, et al. "Never-ending learning". In: *Communications of the ACM* 61.5 (2018), pp. 103–115.

[121] Tom M Mitchell. "Machine Learning". In: Mcgraw hill science, 1997, p. 27.

[122] Tom M Mitchell. *The need for biases in learning generalizations*. Rutgers Univ. New Jersey, 1980.

[123] Tom M Mitchell, John Allen, Prasad Chalasani, John Cheng, Oren Etzioni, Marc Ringuette, and Jeffrey C Schlimmer. "Theo: A framework for self-improving systems". In: *Architectures for intelligence* (1991), pp. 323–355.

[124] Carlos Monserrat, Jose Hernandez-Orallo, Jose-Francisco Dolz, Maria-Jose Ruperez, and Peter Flach. "Knowledge Acquisition by Abduction for Skills Monitoring: Application to Surgical Skills". In: *Inductive Logic Programming*. Springer. 2016.

[125] Michael zur Muehlen and Robert Shapiro. "Business process analytics". In: *Handbook on Business Process Management 2*. Springer, 2010, pp. 137–157.

[126] Florian Mueller and Andrea Lockerd. "Cheese: tracking mouse movement activity on websites, a tool for user modeling". In: *CHI'01 extended abstracts on Human factors in computing systems*. ACM. 2001, pp. 279–280.

[127] Stephen Muggleton. *Inductive logic programming*. 38. Morgan Kaufmann, 1992.

[128] Stephen Muggleton. "Inverse entailment and Progol". In: *New generation computing* 13.3-4 (1995), pp. 245–286.

[129] Stephen Muggleton and Luc De Raedt. "Inductive logic programming: Theory and methods". In: *The Journal of Logic Programming* 19 (1994), pp. 629–679.

[130] Stephen Muggleton, Cao Feng, et al. "Efficient induction of logic programs". In: *Inductive logic programming* 38 (1992), pp. 281–298.

[131] Stephen Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. "Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited". In: *Machine Learning* 100.1 (2015), pp. 49–73.

[132] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[133] Alfredo Nazabal, Christopher KI Williams, Giovanni Colavizza, Camila Rangel Smith, and Angus Williams. "Data Engineering for Data Analytics: A Classification of the Issues, and Case Studies". In: *arXiv preprint arXiv:2004.12929* (2020).

[134] Kan Nishida. *7 Most Practically Useful Operations When Wrangling with Text Data in R*. Sept. 2016. URL: https://blog.exploratory.io/7-most-practically-useful-operations-when-wrangling-with-text-data-in-r-7654bd9d1a0c.

[135] Louis Oliphant and Jude Shavlik. "Using Bayesian networks to direct stochastic search in inductive logic programming". In: *International Conference on Inductive Logic Programming*. Springer. 2007, pp. 191–199.

[136] Roland Olsson. "Inductive functional programming using incremental program transformation". In: *Artificial intelligence* 74.1 (1995), pp. 55–81.

[137] José Hernández Orallo, María José Ramírez Quintana, and César Ferri Ramírez. *Introducción a la Minería de Datos*. Pearson Educación, 2004.

[138] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. "Neuro-symbolic program synthesis". In: *arXiv preprint arXiv:1611.01855* (2016).

[139] D.J. Patil and T. Davenport. "Data scientist: The sexiest job of the 21st century". In: *Harvard business review* 90.10 (2012), pp. 70–76.

[140] Norman W Paton. "Automating Data Preparation: Can We? Should We? Must We?" In: *CEUR Proceedings* (2019).

[141] Simon Peyton-Jones. "Compiling Haskell by program transformation: A report from the trenches". In: *Programming Languages and Systems, ESOP96* (1996), pp. 18–44.

[142] Simon Peyton-Jones, ed. *Haskell 98 Language and Libraries: The Revised Report*. http://haskell.org/, Sept. 2002, p. 277. URL: http://haskell.org/definition/haskell98-report.pdf.

[143] Michael Polanyi. *The tacit dimension*. University of Chicago press, 2009.

[144] Oleksandr Polozov and Sumit Gulwani. "Flashmeta: A framework for inductive program synthesis". In: *ACM SIGPLAN Notices* 50.10 (2015), pp. 107–126.

[145] Oleksandr Polozov and Sumit Gulwani. "Program synthesis in the industrial world: Inductive, incremental, interactive". In: *5th Workshop on Synthesis (SYNT)*. 2016.

[146] Gil Press. "Cleaning big data: Most time-consuming, least enjoyable data science task, survey says". In: *Forbes, March* 23 (2016), p. 15.

[147] J Ross Quinlan. "Learning efficient classification procedures and their application to chess end games". In: *Machine learning*. Springer, 1983, pp. 463–482.

[148] Vijayshankar Raman and Joseph M. Hellerstein. "Potter's Wheel: An Interactive Data Cleaning System". In: *Proceedings of the 27th International Conference on Very Large Data Bases*. VLDB '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 381–390.

[149] Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. "Programming by example using least general generalizations". In: *Twenty-Eighth AAAI Conference on Artificial Intelligence*. 2014.

[150] Emre Rençberoğlu. *Fundamental Techniques of Feature Engineering for Machine Learning*. 2019.

[151] Alan Said and Vicenç Torra. *Data Science in Practice*. Springer, 2019.

[152] Gerard Salton and Christopher Buckley. "Term-weighting approaches in automatic text retrieval". In: *Information Processing 'I&' Management* 24.5 (1988), pp. 513–523. ISSN: 0306-4573.

[153] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://dx.doi.org/10.1016/j.neunet.2014.09.003.

[154] Sara J Shettleworth. *Cognition, evolution, and behavior*. Oxford university press, 2009.

[155] Chengxun Shu and Hongyu Zhang. “Neural Programming by Example.” In: *AAAI*. 2017, pp. 1539–1545.

[156] Rishabh Singh and Sumit Gulwani. “Predicting a correct program in programming by example”. In: *International Conference on Computer Aided Verification*. Springer. 2015, pp. 398–414.

[157] Rishabh Singh and Sumit Gulwani. “Transforming Spreadsheet Data Types Using Examples”. In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’16. St. Petersburg, FL, USA: ACM, 2016, pp. 343–356. ISBN: 978-1-4503-3549-2. DOI: 10.1145/2837614.2837668. URL: http://doi.acm.org/10.1145/2837614.2837668.

[158] Ashwin Srinivasan, Ross D King, and Michael E Bain. “An empirical study of the use of relevance information in inductive logic programming”. In: *JMLR* 4.Jul (2003), pp. 369–383.

[159] European Committee for standarization. “European ICT professionals role profiles - Part 1: 30 ICT profiles”. In: (2018).

[160] Phillip D. Summers. “A Methodology for LISP Program Construction from Examples”. In: *J. ACM* 24.1 (Jan. 1977), pp. 161–175. ISSN: 0004-5411. DOI: 10.1145/321992.322002. URL: https://doi.org/10.1145/321992.322002.

[161] *Supported Data Types - Trifacta Wrangler - Trifacta Documentation*. URL: https://docs.trifacta.com/display/PE/Supported+Data+Types.

[162] *SYNTH Project*. URL: http://synth.cs.kuleuven.be/.

[163] Eindhoven University of Technology. “Process Mining: Data science in Action”. In: (2020). URL: https://www.classcentral.com/course/procmin-2445.

[164] Wayne Thompson, Hui Li, and Alison Bolen. *Artificial intelligence, machine learning, deep learning and beyond*. 2020.

[165] Alan Turing. “Computing machinery and intelligence”. In: *Mind* 59.236 (1950), p. 433.

[166] Wil MP Van der Aalst. “Data scientist: The engineer of the future”. In: *Enterprise Interoperability VI*. Springer, 2014, pp. 13–26.

[167] Alta Van der Merwe, Aurona Gerber, and Hanlie Smuts. “Guidelines for Conducting Design Science Research in Information Systems”. In: *Annual Conference of the Southern African Computer Lecturers’ Association*. Springer. 2019, pp. 163–178.

[168] Alejandro Vera-Baquero, Ricardo Colomo-Palacios, and Owen Molloy. "Business process analytics using a big data approach". In: *IT Professional* 15.6 (2013), pp. 29–35.

[169] Ruben Verborgh and Max De Wilde. *Using OpenRefine*. Packt Publishing Ltd, 2013.

[170] Analytics Vidhya. *19 Data Science Tools for people who aren't so good at Programming*. May 2016. URL: https://www.analyticsvidhya.com/blog/2016/05/19-data-science-tools-for-people-dont-understand-coding/.

[171] Xinyu Wang, Isil Dillig, and Rishabh Singh. "Program synthesis using abstraction refinement". In: *Proceedings of the ACM on Programming Languages* 2.POPL (2017), p. 63.

[172] *Wrangle Language - Trifacta Wrangler - Trifacta Documentation*. URL: https://docs.trifacta.com/display/PE/Wrangle+Language.

Appendices

Appendix A

Data Collection

A.1 Survey to collect personal data

Automating the data cleansing process

Hi! My name is Lidia and I'm a PhD student at the Universitat Politècnica de València (Spain). My research is focused on the data cleansing automation. However, achieving this goal requires a lot of data. Can you help me?

The purpose of this form is to collect personal data (names, dates, emails, addresses, etc.). They don't have to be real, we just need the correct format (having a '@' if it is an email, for example). You can fill it as many times as you want.

By submitting this form you are indicating that you have read the description of the study, are over the age of 18, and that you agree to the terms as described.

Thank you! :)

1. **I agree to participate in the research study. I understand the purpose and nature of this study and I am participating voluntarily. I understand that I can withdraw from the study at any time, without any penalty or consequences.**
   - Yes
   - No
2. **Name**
   It can be a composed name.
3. **Surname**
   It can be a composed surname.

4. **Title**
   Title prefixing the person's name (e.g. Miss, Ms, Mr, etc.).
5. **Gender**
   - Man
   - Woman
   - Other
6. **Email**
   An email address containing at least an '@' and a dot. It can contain other symbols (e.g. my-email@my-email.com).
7. **Address**
   Use the style you would normally use.
8. **Phone number**
   Use the style you would normally use.
9. **Date**
   Use the style you would normally use.
10. **Date (standard)**
    Same date as above with a provided calendar.
11. **Hour** *Use the style you would normally use.*
12. **Hour (standard)**
    Same hour as above with a provided (digital) clock.
13. **Comments**
    Do you want to leave a comment?
14. **Country**
    Where are you from? (Your real country).

Appendix B

Data Wrangling: Competences and Skills

Data scientist has been classified as "the sexiest job of the 21st century" [139]. However, as we have seen in the previous sections, data science is a process that follows several and very different phases and, each of them, requires different competences and skills to be done properly. So, being a data scientist implies to be able to apply techniques from very different fields. Overall, data scientists should have a T-shaped profile [57], that is, they need deep expertise in the field and, at the same time, very different interdisciplinary skills and abilities to collaborate with others in different areas. This means not only possessing deep, technical skills, but also having broader attributes such as empathy or communication skills.

In a data science project one or more data scientists can take part. When working alone, a data scientist needs a list competences in order to achieve its goals. When one or more data scientists are working in a data science team each of them can be a specialist in one or more phases of the process and, therefore, has only or more developed the necessary competences for that particular phase. Some competences have been identified as necessary for any data scientists in several studies, including both technical and non-technical skills [1, 37, 108]. The European e-Competence Framework (e-CF) [43] provides a standard reference for skills and competences related with Information and Communication Technology (ICT). According to this standard a competence is a "demonstrated ability to apply knowledge, skills and attitudes for achieving observable results". e-CF classify the competences in five different proficiency levels according to five main ICT areas (Plan, Build, Run, Enable and Manage) and related to the European Qualifications Framework (EQF) [16]. However, as Figure B.1 shows, in order to reach these competences several skills are required including technical knowledge for 'hard-skills' and personal attitudes for 'soft-skills'.

The e-CF standard [34] report on data science profile competences describes a data scientist as the person that: (1) leads the process of applying data analytics; (2) delivers insights from data presenting visual data representations, and (3) finds, manages and merges multiple data sources and ensures consistency of datasets. Identifies the mathematical models, selects and optimises the algorithms to deliver business value through insights. Communicates patterns and recommends ways of applying data". For this job, the report identifies five main competence groups: (1) technology and trend monitoring; (2) innovating; (3) information and knowledge management; (4) needs identification; and (5) forecast development [159].

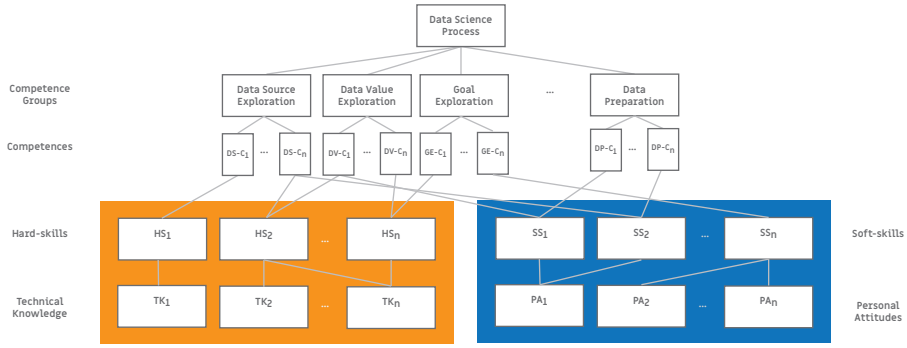


Figure B.1. General schema of competence groups and skills. The diagram is just an example on how the competences can be organised.

B.1 List of Data Wrangling Competences

Here we extend the competence groups to identify the main competences and skills required in data wrangling process. Table B.1 shows the competences identified for this data science step.

Competence	Description
C-DP01	Data integration from multiples sources and structures combining by merging two data sets with similar records but different attributes or appending two or more data sets with similar attributes but different records.
C-DP02	Clean data and fix errors (outliers, missing values, etc.).
C-DP03	Define meta-data using common standards and practices.
C-DP04	Use transformations in datasets to get the identified relevant attributes.
C-DP05	Select data with feature extraction, resolution change and dimensionality reduction techniques to define possible attribute sets for modelling activities.
C-DP06	Generate new attributes by derived attributes or entire new records, or transformed values for existing attributes.
C-DP07	Make syntactic changes in data to satisfy the requirements of the specific modelling tool/project.

Table B.1. Competences required for a data scientists doing data preparation/wrangling.

B.2 List of Data Wrangling Hard Skills

Hard skills are abilities that can be learnt through training, evaluated and accredited. Some typical hard skills are languages or programming. Hard-skills need technical knowledge and can be probed by certificates and provable professional experience. Table B.2 shows some of the hard-skills identified as necessary for data wrangling and Table B.3 the necessary technical knowledge according to this skills.

B.3 List of Data Wrangling Soft Skills

Finally, soft skills or personal attitudes are the cognitive and relational capacity of human-beings [43]. Unlike hard-skills that can be learned, soft skills require personal attitudes inherent to them and characterise how people interact with the environment and with other people. Table B.4 shows some of the soft skills identified as necessary for data wrangling and B.5 their associated personal attitudes.

Automating the whole data wrangling process would require a system to be able to simulate and carry on all the competences, learning the hard skills and, somehow, acquiring the soft skills and the data scientist behaviour. While full automation seems improbable right now since some of the skills are not automatable, semi-automation or the automation of some of the data wrangling phases could be easier.

Skill	Description
HS01	Data Mining: text mining, anomaly detection, time series, feature selection, etc.
HS02	Qualitative analytics.
HS03	Natural language processing.
HS04	Data preparation and pre-processing.
HS05	Data anonymisation.
HS06	Information systems, collaborative systems.
HS07	Data architecture, data types and data formats.
HS08	Data curation and data quality, data integration and interoperability.
HS09	Metadata, data registries, data factories.
HS10	Data collection and data quality evaluation.
HS11	Emerging technologies.
HS12	Relevant sources of information.
HS13	Applied research programme approaches.
HS14	Habits, trends and needs.
HS15	Innovation processes techniques.

Table B.2. Hard skills identified in data wrangling process

Knowledge	Description
TK01	Statistical computing and languages (R, Python, SAS, Julia, WEKA, KNIME, etc.).
TK02	Matlab data analytics.
TK03	Analytics tools (RStudio, Anaconda, SPSS, Matlab, etc.).
TK04	Data mining tools (RapidMiner, Orange, etc.).
TK5	Data curation platform, metadata management (ETL, Curato's Workbench, DataUp, etc.).

Table B.3. Technical knowledge identified to carry on the hard skills on data wrangling.

Skill	Description
SK01	Make decisions.
SK02	Understand results.
SK03	Recognise patterns.
SK04	Making inferences.
SK05	Communication and understanding natural language.
SK06	Create/design.
SK07	Keeping a to-do list.
SK08	Online learning.
SK09	Continuous learning/improvement.
SK10	Collaboration with other team members, other roles (multidisciplinary team), other teams (Universities, companies, etc.).
SK11	Domain fundamentals (knowledge, iterative development, goal).
SK12	Know limitations of technology.
SK13	Understand visualisations.
SK14	Know when something fails.
SK15	Communication with experts/non-experts in the domain.
SK16	Ability to answer/ask technical/domain questions.
SK17	Acquire/improve knowledge about the domain.
SK18	Technical problem-solution.
SK19	Use professional networks and online communities to learn/ask.
SK20	Search/read/learn from online domain resources (papers, news, wikipedia, etc.).
SK21	Accept/fix fails in problem-domain.
SK22	Adaptation to technology changes and updates.
SK23	Innovate.
SK24	Invent.
SK25	Think out of the box.
SK26	Identify appropriate resources.

Table B.4. Soft skills identified in data wrangling process.

Attitude	Description
PA01	Critical thinking.
PA02	Natural language (writing/speaking/reading/understanding).
PA03	Creativity.
PA04	Planning & organising (time management).
PA05	Dynamic (self) re-skilling.
PA06	Ability to collaborate.

Table B.5. Personal attitudes identified to carry on the soft skills on data wrangling.

Appendix C

Logging Data Scientists

If we really want to automate data science, and data wrangling in particular, we need to know how data scientists behave. In other words, we have to apply data science to data scientists (these ideas have been published at [26]). However, it seems very difficult to track all the activities a data scientist (or a data science team) is doing. Indeed, apart from a few surveys (about the tools and times they devote to every stage of the whole process), there is a lack of evidence about what data scientists really do and the decisions and actions they take, especially at a high granularity level. The introduction of data mining tools in the past two decades, such as SPSS Clementine (then IBM Modeler), Weka KnowledgeFlow, SAS Enterprise Miner, RapidMiner and many others that followed, made it possible, for the first time, to incorporate most of a data science process into the same tool. However, logging the actions of the users had to be done locally, with the difficulty of obtaining a relative good number of expert experiences. Collaborative or competitive platforms such as Kaggle or Github can also be a source of data, but it is difficult to extract information about sequential workflow or the particular actions that have to be taken for all the stages of a data science project. This is aggravated by the recent "back to programming" trend, where the products of data scientists in these platforms are programs (usually in R or python), and not a sequence of actions over a structured set of possibilities. In fact, some tools that try to automate the process are based on the "knowledge, experience and best practices" of data scientists, such as DataRobot, but not based on the evidence of real logs at a high granularity level.

Things are different for cloud data science tools, and many old platforms are migrating or are native there, such as BigML[1], DataRobot[2], Azure ML[3], ClowdFlows[4] and others [170]. In these platforms we can log the activity of data scientists and use that activity to recommend actions according to the interactions of many data scientists on the same platforms for similar situations.

Many scientists working on data wrangling, story telling or visualisation, may not be experts or even comfortable with computer programming. In order to simplify these tasks and to make them easy to the maximum number of people, there exist some tools that reduce the data science process into a few visual steps, that can be done with not many clicks and without any kind of programming.

In order to automate each part of the data science process (including data wrangling), the study of the use of this kind of tools can be very useful, since platforms can store all the workflows created and in some of them we can also

[1] https://bigml.com/
[2] https://www.datarobot.com/
[3] https://azure.microsoft.com/en-us/services/machine-learning/
[4] http://www.clowdflows.org/

look over the steps followed in the process, from the upload of datasets until the obtained results (including data cleaning process).

Tracking user behaviour through software interaction is a common topic in areas such as web usability (by the use of technologies involving mouse or eye tracking) [56, 126] and business intelligence (using behavioural analytics) [125, 168], even including some extra context data of users (such as demographic data) [94]. Activity logging is also common in ambient intelligence from fixed sensors or from mobile devices [113]. Nevertheless, the goal of tracking and exploiting non-trivial operational processes is represented by the area of *process mining* [2]. Process mining seeks to process "event logs" (information about business processes stored by information systems) so as to discover, monitor and improve processes (i.e., check the conformance of processes, detect bottlenecks or predict execution problems) by means of process analytics. However, to our knowledge, process mining has not been applied to the data science process itself.

Process mining bridges the gap between traditional model-based process analysis (e.g., simulation and other business process management techniques) and data-centric analysis techniques such as machine learning and data mining. Process mining seeks the confrontation between event data (i.e., observed behaviour) and process models (hand-made or discovered automatically). This technology has become available only recently, but it can be applied to any type of operational processes (organisations and systems). Example applications include: analysing treatment processes in hospitals, improving customer service processes in a multinational, understanding the browsing behaviour of customers using booking site, analysing failures of a baggage handling system, and improving the user interface of an X-ray machine. All of these applications have in common that dynamic behaviour needs to be related to process models. Hence, we refer to this as "data science in action" [163].

The first thing we can analyse is what to log and how to represent the events in data science tools such that we are able to track the full data science process. It is important that we distinguish the data or knowledge flow, as represented by many graphical data science tools (see Figure C.1, bottom) from the log of actions and events (the process) that led to that flow. It is especially important that we can track mistakes, trials and other attempted actions that are not finally represented by the flow, and this can only be done from the log. Figure C.1 (top) shows the procedure of extracting the events from Clowdflows (a visual data science tool, [99]) including fixed attributes (timestamp, user, resource, transaction type, etc.) and event-specific ones (node identifiers, parameters, ports, inputs, outputs, etc.).

Events refer to actions and properties performed by a data scientist in an specific data science tool which can be traced back to generate the complete workflow. Events may have any number of attribute, including standard (timestamp, user, resource, transaction type, etc.) and action-specific ones (node/s identifier/s, parameters, ports, inputs, outputs, etc.). Although timestamps could be used to align events in the horizontal dimension, it could be also possible to align events based on the activities performed, so it is easy to identify common behaviours and deviations.

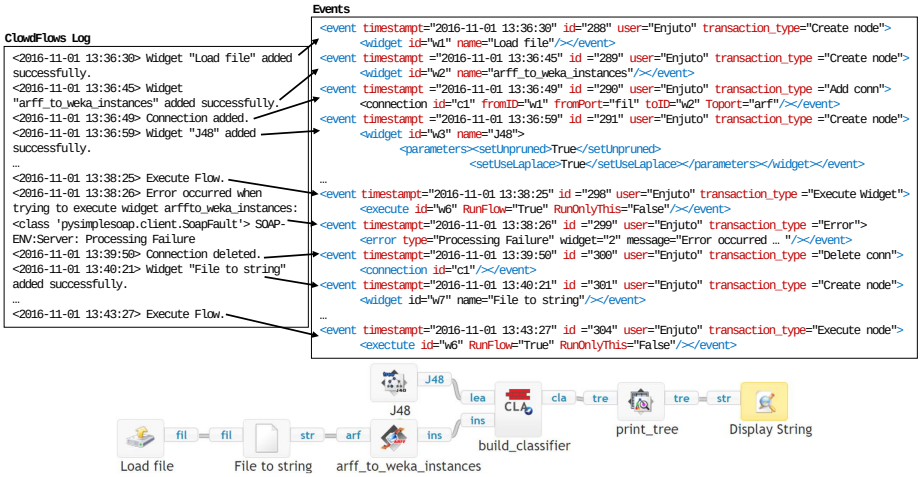


Figure C.1. Top: example of a log using Clowdflows and its formatting into events. Bottom: the corresponding data-to-knowledge flow finally completed by the data scientist.

The second issue to consider is how to analyse the data from the formatted events and the flow. As the tracked events are at a very low-level (e.g., connect a node X with Y, execute node Y), we have to use abstraction in order to match series of them with more high-level events (e.g., perform feature selection). There are several approaches to activity recognition in the literature, but we are considering two different approaches: a logical approach using event calculus, as done by [124], where processes can be matched with given instructions (e.g., an exercise given to a student or a data scientist), and a reinforcement learning approach, where the set of possible actions at each point is limited by the use of contextual information, where repetitive tasks can be spotted.

The extent and possibilities of this analysis and the use of other AI tools depend on the particular application, what parts of the pipeline are to be analysed and the understandability of the insight. As we can gather data science processing information from the use of Clowdflows (and perhaps in the future from some other tools, such as BigML), we are encouraging expert data scientists, practitioners and students to be logged, so that we can use all this information for a better understanding of the data science process, common mistakes and recipes for success. In terms of automation, the introduction of assistants in the same tools can be a first step, but we envisage some other possibilities along the lines of some of the ongoing initiatives for the automation of data science: Chalearn's AutoML [64], the Automatic Statistician [105] and the SYNTH project [162].

Overall, the generation of this logged information as open source data can be very useful for the data science community in general, but especially useful for the application of AI to data science and ultimately for the (semi-)automation of data science (and hence, data wrangling) in a more holistic way.

Appendix D

BK-ADAPT: Supplementary Material

D.1 Background Knowledge: List of Functions

D.1.1 Default BK: MagicHaskeller's library

By default, *MagicHaskeller* includes a list of 189 basic Haskell functions:

id	Function
001	0 :: Int
002	1 :: Int
003	(++) :: forall a . (->) ([a]) ([a] -> [a])
004	filter :: forall a . (a -> Bool) -> [a] -> [a]
005	negate :: Ratio Int -> Ratio Int
006	abs :: Ratio Int -> Ratio Int
007	sum :: (->) ([Ratio Int]) (Ratio Int)
008	product :: (->) ([Ratio Int]) (Ratio Int)
009	(+) :: Ratio Int -> Ratio Int -> Ratio Int
010	(-) :: Ratio Int -> Ratio Int -> Ratio Int
011	(*) :: Ratio Int -> Ratio Int -> Ratio Int
012	(/) :: Ratio Int -> Ratio Int -> Ratio Int
013	fromIntegral :: Int -> Ratio Int
014	properFraction :: (->) (Ratio Int) ((Int, Ratio Int))
015	round :: (->) (Ratio Int) Int
016	floor :: (->) (Ratio Int) Int
017	ceiling :: (->) (Ratio Int) Int
018	(i :: Ratio Int -> Int -> Ratio Int
019	(%) :: Int -> Int -> Ratio Int
020	numerator :: (->) (Ratio Int) Int
021	denominator :: (->) (Ratio Int) Int
022	[] :: forall a . [a]
023	(:) :: forall a . a -> [a] -> [a]
024	foldr :: (a -> b -> b) -> b -> [a] -> b
025	drop 1 :: Int -> [a] -> [a]
026	(+) :: Int -> Int
027	n x f -> iterate f x !! (n::Int)
028	Nothing :: forall a . Maybe a
029	Just :: forall a . a -> Maybe a

continued on next page

continued from previous page

id	Function
030	maybe :: a -> (b -> a) -> Maybe b -> a
031	True :: Bool
032	False :: Bool
033	iF :: forall a . (->) Bool (a -> a -> a)
034	(+) :: (->) Int ((->) Int Int)
035	(&&) :: (->) Bool ((->) Bool Bool)
036	(||) :: (->) Bool ((->) Bool Bool)
037	not :: (->) Bool Bool
038	(-) :: Int -> Int -> Int
039	(*) :: Int -> Int -> Int
040	map :: (a -> b) -> [a] -> [b]
041	concatMap :: (a -> [b]) -> [a] -> [b]
042	length :: forall a . (->) ([a]) Int
043	replicate :: forall a . Int -> a -> [a]
044	take :: forall a . Int -> [a] -> [a]
045	drop :: forall a . Int -> [a] -> [a]
046	takeWhile :: forall a . (a -> Bool) -> [a] -> [a]
047	dropWhile :: forall a . (a -> Bool) -> [a] -> [a]
048	reverse :: forall a . [a] -> [a]
049	and :: (->) ([Bool]) Bool
050	or :: (->) ([Bool]) Bool
051	any :: (a -> Bool) -> [a] -> Bool
052	all
053	zipWith
054	null :: forall a . (->) ([a]) Bool
055	abs :: (->) Int Int
056	foldl
057	total head
058	total last
059	total init
060	enumFromTo :: Int -> Int -> [Int]
061	enumFromTo :: Char -> Char -> [Char]
062	fmap :: forall a b . (a -> b) -> (->) (Maybe a) (Maybe b)
063	either
064	gcd :: Int -> Int -> Int
065	lcm :: Int -> Int -> Int
066	sum :: (->) ([Int]) Int
067	product :: (->) ([Int]) Int
068	(==)
069	(/=)
070	compare
071	(<=)

continued on next page

continued from previous page

id	Function
072	(<)
073	max
074	min
075	sortBy :: forall a . (a -> a -> Ordering) -> [a] -> [a]
076	nubBy :: forall a . (a -> a -> Bool) -> [a] -> [a]
077	deleteBy :: forall a . (a -> a -> Bool) -> a -> [a] -> [a]
078	dropWhileEnd :: forall a . (a -> Bool) -> [a] -> [a]
079	transpose :: forall a . [[a]] -> [[a]]
080	toUpper :: (->) Char Char
081	toLower :: (->) Char Char
082	ord :: Char -> Int
083	isControl :: (->) Char Bool
084	isSpace :: (->) Char Bool
085	isLower :: (->) Char Bool
086	isUpper :: (->) Char Bool
087	isAlpha :: (->) Char Bool
088	isAlphaNum :: (->) Char Bool
089	isDigit :: (->) Char Bool
090	isSymbol :: (->) Char Bool
091	isPunctuation :: (->) Char Bool
092	isPrint :: (->) Char Bool
093	10 :: Int
094	20 :: Int
095	30 :: Int
096	40 :: Int
097	' ' :: Char
098	1 :: Double
099	10 :: Double
100	100 :: Double
101	1000 :: Double
102	succ :: Double -> Double
103	negate :: Double -> Double
104	abs :: Double -> Double
105	signum :: Double -> Double
106	recip :: Double -> Double
107	sum :: (->) ([Double]) Double
108	product :: (->) ([Double]) Double
109	(+) :: Double -> Double -> Double
110	(-) :: Double -> Double -> Double
111	(*) :: Double -> Double -> Double
112	(/) :: Double -> Double -> Double
113	fromIntegral :: Int -> Double

continued on next page

continued from previous page

id	Function
114	properFraction :: (->) Double ((Int, Double))
115	round :: (->) Double Int
116	floor :: (->) Double Int
117	ceiling :: (->) Double Int
118	truncate :: (->) Double Int
119	(i :: Double -> Int -> Double
120	pi :: Double
121	lines :: [Char] -> [[Char]]
122	words :: [Char] -> [[Char]]
123	unlines :: [[Char]] -> [Char]
124	unwords :: [[Char]] -> [Char]
125	scanl :: forall a b . (a -> b -> a) -> a -> [b] -> [a]
126	scanr :: forall a b . (a -> b -> b) -> b -> [a] -> [b]
127	scanl:: forall a . (a -> a -> a) -> [a] -> [a]
128	scanr:: forall a . (a -> a -> a) -> [a] -> [a]
129	show :: Int -> [Char]
130	(,) :: forall a b . a -> b -> (a, b)
131	uncurry
132	elem
133	nub
134	find :: forall a . (a -> Bool) -> [a] -> Maybe a
135	findIndex
136	findIndices
137	deleteFirstsBy :: forall a . (a -> a -> Bool) -> [a] -> [a] -> [a]
138	unionBy :: forall a . (a -> a -> Bool) -> (->) ([a]) ([a] -> [a])
139	intersectBy :: forall a . (a -> a -> Bool) -> (->) ([a]) ([a] -> [a])
140	insertBy :: forall a . (a -> a -> Ordering) -> a -> [a] -> [a]
141	isOctDigit :: (->) Char Bool
142	isHexDigit :: (->) Char Bool
143	catMaybes :: forall a . [Maybe a] -> [a]
144	listToMaybe :: forall a . (->) ([a]) (Maybe a)
145	maybeToList :: forall a . (->) (Maybe a) ([a])
146	exp :: Double -> Double
147	log :: Double -> Double
148	sqrt :: Double -> Double
149	(**) :: Double -> Double -> Double
150	logBase :: Double -> Double -> Double
151	sin :: Double -> Double
152	cos :: Double -> Double
153	tan :: Double -> Double
154	asin :: Double -> Double
155	acos :: Double -> Double

continued on next page

continued from previous page

id	Function
156	atan :: Double -> Double
157	sinh :: Double -> Double
158	cosh :: Double -> Double
159	tanh :: Double -> Double
160	asinh :: Double -> Double
161	acosh :: Double -> Double
162	atanh :: Double -> Double
163	floatDigits :: Double -> Int
164	exponent :: Double -> Int
165	significand :: Double -> Double
166	scaleFloat :: Int -> Double -> Double
167	atan2 :: Double -> Double -> Double
168	(„) :: forall a b c . a -> b -> c -> (a, b, c)
169	Left :: forall a b . a -> Either a b
170	Right :: forall b a . b -> Either a b
171	zip :: forall a b . (->) ([a]) ((->) ([b]) ([(a, b)]))
172	zip3 :: forall a b c . (->) ([a]) ((->) ([b]) ((->) ([c]) ([(a, b, c)])))
173	unzip :: forall a b . (->) ([(a, b)]) (([a], [b]))
174	unzip3 :: forall a b c . (->) ([(a, b, c)]) (([a], [b], [c]))
175	odd :: Int -> Bool
176	even :: Int -> Bool
177	isPrefixOf
178	isSuffixOf
179	isInfixOf
180	stripPrefix
181	lookup
182	sort
183	intersperse :: forall a . a -> [a] -> [a]
184	subsequences :: forall a . [a] -> [[a]]
185	permutations :: forall a . [a] -> [[a]]
186	inits :: forall a . [a] -> [[a]]
187	tails :: forall a . [a] -> [[a]]
188	mapAccumL
189	mapAccumR

Table D.1. Functions included by default in *MagiccHaskeller*.

D.1.2 Freetext: Basic string manipulation functions

id	Function
001	2 :: Int
002	3 :: Int

continued on next page

continued from previous page

id	Function
003	4 :: Int
004	5 :: Int
005	6 :: Int
006	7 :: Int
007	8 :: Int
008	9 :: Int
009	11::Int
010	12::Int
011	13::Int
012	14::Int
013	15::Int
014	16::Int
015	17::Int
016	18::Int
017	19::Int
018	21::Int
019	22::Int
020	23::Int
021	24::Int
022	25::Int
023	26::Int
024	27::Int
025	28::Int
026	29::Int
027	31::Int
028	1900::Int
029	2000::Int
030	dash :: [Char]
031	slash :: [Char]
032	dot :: [Char]
033	comma :: [Char]
034	colon :: [Char]
035	lBracket :: [Char]
036	rBracket :: [Char]
037	at :: [Char]
038	hash :: [Char]
039	lparentheses :: [Char]
040	rparentheses :: [Char]
041	space :: [Char]
042	zero :: [Char]
043	nineteen :: [Char]
044	twenty :: [Char]

continued on next page

continued from previous page

id	Function
045	firstElement :: [Char]
046	middleElement :: [Char]
047	lastElement :: [Char]
048	addPunctuationString :: [Char] -> [Char] -> [[Char]]
049	splitStringWithoutPunctuation :: [Char] -> [[Char]]
050	setPunctuationArray :: [[Char]] -> [Char] -> [[Char]]
051	changePunctuationArray :: [[Char]] -> [Char] -> [[Char]]
052	changePunctuationString :: [Char] -> [Char] -> [Char]
053	deletePunctuationArray :: [[Char]] -> [[Char]]
054	deletePunctuationString :: [Char] -> [Char]
055	deleteSomePunctuationString :: [Char] -> [Char] -> [Char]
056	splitStringByPunctuation :: [Char] -> [Char] -> [[Char]]
057	splitStringWithPunctuation :: [Char] -> [[Char]]
058	splitStringTakeOffPunctuation :: [Char] -> [[Char]]
059	swapElementsString :: Int -> Int -> [Char] -> [Char]
060	swapElementsArray :: Int -> Int -> [[Char]] -> [[Char]]
061	appendPositionArray :: [[Char]] -> [Char] -> Int -> [[Char]]
062	appendPositionString :: [Char] -> [Char] -> Int -> [Char]
063	appendNextToLast :: [[Char]] -> [Char] -> [[Char]]
064	append :: [Char] -> [Char] -> [Char]
065	append_first :: [[Char]] -> [Char] -> [[Char]]
066	append_middle :: [[Char]] -> [Char] -> [[Char]]
067	append_last :: [[Char]] -> [Char] -> [[Char]]
068	prepend :: [Char] -> [Char] -> [Char]
069	prepend_first :: [[Char]] -> [Char] -> [[Char]]
070	prepend_middle :: [[Char]] -> [Char] -> [[Char]]
071	prepend_last :: [[Char]] -> [Char] -> [[Char]]
072	replacePositionArray :: [[Char]] -> [Char] -> Int -> [[Char]]
073	replacePositionString :: [Char] -> [Char] -> Int -> [Char]
074	replacePositionArrayFixedSize :: [[Char]] -> [Char] -> [Char] -> [[Char]]
075	replaceAll :: [Char] -> [Char] -> [Char] -> [Char]
076	replaceNextToLast :: [[Char]] -> [Char] -> [[Char]]
077	toLowString :: [Char] -> [Char]
078	toUpperString :: [Char] -> [Char]
079	reduceWord :: [Char] -> Int -> [Char]
080	takeOneOfArray :: [[Char]] -> Int -> [Char]
081	takeOneOfFixedSizeArray :: [[Char]] -> [Char] -> [Char]
082	takeOneOfFixedSizeString :: [Char] -> [Char] -> [Char]
083	joinStringsWithPunctuation :: [Char] -> [Char] -> [Char] -> [Char]

continued on next page

continued from previous page

id	Function
084	getOneWordByPosition :: [Char] -> Int -> [Char]
085	getFirstWord :: [Char] -> [Char]
086	getLastWord :: [Char] -> [Char]
087	getOneCharacterByPosition :: [Char] -> Int -> [Char]
088	getFirstCharacter :: [Char] -> [Char]
089	getLastCharacter :: [Char] -> [Char]
090	getStartToFirstSymbolOccurrence :: [Char] -> [Char] -> [Char]
091	getStartToLastSymbolOccurrence :: [Char] -> [Char] -> [Char]
092	getLastSymbolOccurrenceToEnd :: [Char] -> [Char] -> [Char]
093	getFirstSymbolOccurrenceToEnd :: [Char] -> [Char] -> [Char]
094	joinArrayWithPunctuation :: [[Char]] -> [Char] -> [Char]
095	joinStringsWithoutPunctuation :: [Char] -> [Char] -> [Char]
096	setParentheses :: [Char] -> [Char]
097	getCaps :: [Char] -> [Char]
098	reduceSpaces :: [Char] -> [Char]
099	setBrackets :: [Char] -> [Char]
100	completeBrackets :: [Char] -> [Char]
101	completeParentheses :: [Char] -> [Char]
102	getFirstDigitToEnd :: [Char] -> [Char]
103	getStartToFirstDigit :: [Char] -> [Char]
104	insert_first :: [[Char]] -> [Char] -> [[Char]]
105	insert_last :: [[Char]] -> [Char] -> [[Char]]
106	deleteParentheses :: [Char] -> [Char]
107	changeSomePunctuationString :: [Char] -> [Char] -> [Char] -> [Char]
108	removeWords :: [Char] -> [[Char]] -> [Char]

Table D.2. Functions generated for the *Freetext* domain.

D.1.3 Domain Functions

D.1.3.1 Dates

id	Function
01	getDayCardinalString::[Char]->[Char]
02	getDayCardinalArray::[[Char]]->[Char]
03	getDayOrdinal::[Char]->[Char]
04	getWeekDayArray::[[Char]]->[Char]
05	getWeekDayString::[Char]->[Char]
06	getMonthNameString::[Char]->[Char]
07	getMonthNameArray::[[Char]]->[Char]
08	convertMonth::[Char]->[Char]

continued on next page

continued from previous page

id	Function
09	getYearString::[Char]->[Char]
10	getYearArray::[[Char]]->[Char]
11	convertMonthToNumeric::[Char]->[Char]
12	convertMonthToString::[Char]->[Char]
13	takeTwoOfThreeArray::[[Char]]->Int->Int->[[Char]]
14	getMonthArray::[[Char]]->[Char]
15	getMonthString::[Char]->[Char]
16	convertMonthToNumericWithinArray::[[Char]]->[[Char]]
17	convertMonthToStringWithinArray::[[Char]]->[[Char]]
18	reduceMonthWithinArray::[[Char]]->[[Char]]
19	changeDateFormat::[Char]->[Char]->[[Char]]
20	convertDayOrdinalWithinArray::[[Char]]->[[Char]]
21	reduceYear::[Char]->[Char]
22	reduceYearWithinArray::[[Char]]->[[Char]]
23	reduceMonth::[Char]->[Char]

Table D.3. Functions generated for the *Dates* domain.

D.1.3.2 Emails

id	Function
1	getWordsBeforeAt :: [Char] -> [Char]
2	getWordsAfterAt :: [Char] -> [Char]
3	getWordsBeforeDot :: [Char] -> [Char]
4	getWordsAfterDot :: [Char] -> [Char]
5	getWordsBetweenAtAndDot :: [Char] -> [Char]
6	appendAt :: [Char] -> [Char]
7	prependAt :: [Char] -> [Char]
8	joinStringsWithAt :: [Char] -> [Char] -> [Char]
9	dotcom :: [Char]

Table D.4. Functions generated for the *Emails* domain.

D.1.3.3 Names

id	Function
01	addMaleNomenclature :: [Char] -> Int -> [Char]
02	addFemaleNomenclature :: [Char] -> Int -> [Char]
03	deleteNomenclature :: [Char] -> [Char]
04	getNomenclature :: [Char] -> [Char]
05	reduceNameSecondWord :: [Char] -> [Char] -> [Char]
06	getGenderByNomenclature :: [Char] -> [Char]
07	deleteNomenclatureAndPunctuation :: [Char] -> [Char]

continued on next page

continued from previous page

id	Function
08	reduceNamesFirstPlace :: [Char] -> [Char]
09	reduceNameFirstPlace :: [Char] -> [Char]
10	reduceNameWithSurnameSecondPlace :: [Char] -> [Char]
11	reduceNameWithSurnamesSecondPlace :: [Char] -> [Char]
12	initialsNameFirstPlace :: [Char] -> [Char]

Table D.5. Functions generated for the *Names* domain.

D.1.3.4 Phones

id	Function
1	addPhonePrefix :: [Char] -> Int -> [Char]
2	addPhonePrefixByCountry :: [Char] -> [Char] -> [Char]
3	addPhonePrefixByCountryCode :: [Char] -> [Char] -> [Char]
4	addPlusInPrefix :: [Char] -> [Char]
5	getPhoneNumber :: [Char] -> [Char]

Table D.6. Functions generated for the *Phones* domain.

D.1.3.5 Times

id	Function
01	integerToTime :: Int -> [Char]
02	appendTimeElement :: [Char] -> Int -> [Char]
03	convertTimeByTimeZone :: [Char] -> [Char] -> [Char] -> [Char]
04	getTime :: [Char] -> [Char]
05	getHour :: [Char] -> [Char]
06	getMinutes :: [Char] -> [Char]
07	getSeconds :: [Char] -> [Char]
08	appendOclockTime :: [Char] -> [Char]
09	appendSomeTime :: [Char] -> Int -> [Char]
10	changeHour :: [Char] -> Int -> [Char]
11	changeMinutes :: [Char] -> Int -> [Char]
12	changeSeconds :: [Char] -> Int -> [Char]
13	deleteLastTimePosition :: [Char] -> [Char]
14	increaseHour :: [Char] -> Int -> [Char]
15	decreaseHour :: [Char] -> Int -> [Char]
16	increaseMinutes :: [Char] -> Int -> [Char]
17	decreaseMinutes :: [Char] -> Int -> [Char]
18	increaseSeconds :: [Char] -> Int -> [Char]
19	decreaseSeconds :: [Char] -> Int -> [Char]
20	convertTimeTo24hoursFormat :: [Char] -> [Char]
21	convertTimeTo12hoursFormat :: [Char] -> [Char]

continued on next page

continued from previous page

id	Function
22	convertTimeFormat :: [Char] -> [Char] -> [Char]
23	get12hoursFormatAuxiliar :: [Char] -> [Char]
24	delete12hoursFormatAuxiliar :: [Char] -> [Char]

Table D.7. Functions generated for the *Times* domain.

D.1.3.6 Units

id	Function
1	unitsConversion :: Float -> [Char] -> [Char] -> Float
2	getUnits :: [Char] -> [Char]
3	getSystem :: [Char] -> [Char]
4	setUnits :: Float -> [Char] -> [Char]

Table D.8. Functions generated for the *Units* domain.

D.2 Data: List of Meta-features

In order to describe different characteristics of the inputs and use them with the domain classifier and function ranker, we have defined the following descriptive *meta-features* that can be extracted automatically:

id	Meta-feature
01	end_digit
02	end_dotAndWord
03	end_lower
04	end_upper
05	has_1at
06	has_1comma
07	has_2blank
08	has_2colon
09	has_2dash
10	has_2digits
11	has_2dot
12	has_2slash
13	has_4digits
14	has_6digits
15	has_8digits
16	has_9OrMoredigits
17	has_at
18	has_blanks
19	has_capitalAndDot
20	has_colon
21	has_courtesyTitles
22	has_dash
23	has_dayName
24	has_dot
25	has_hourStructure
26	has_hoursWords
27	has_lettersAndDot
28	has_lowers
29	has_monthName
30	has_numbers
31	has_numbersInsideParenthesis
32	has_only1Word
33	has_only2Words
34	has_only3Words
35	has_onlyNumbersAndSymbols
36	has_ordinalNumbers
37	has_plus

continued on next page

continued from previous page

id	Meta-feature
38	has_punctuation
39	has_slash
40	has_unitsSystem
41	has_uppers
42	has_wordAndComma
43	has_wordsJointByDash
44	is_emailStructure
45	is_empty
46	is_LongDateStructure
47	is_NA
48	is_onlyAlphabetic
49	is_onlyNumeric
50	is_onlyPunctuation
51	is_shortDateStructure
52	start_digit
53	start_lower
54	start_upper

Table D.9. Meta-features generated to characterise the problems from the different domains.

D.3 Experiments: Extended Results

D.3.1 Primitive Estimator

domain	auc
dates	0.92
emails	0.96
names	1
phones	0.97
times	1
units	1

Table D.10. Results for the primitive estimator.

D.3.2 Function Comparison

We have compared the performance of our approach using the ranking strategy with other DSL data wrangling tools, concretely *Trifacta Wrangler* and *FlashFill*. Flashfill works in the same way as our approach, namely, it uses one or more input instances to try to induce a potential solution which is then applied to the rest of examples. If no solution is found or the problem at hand is not solvable by Flashfill, it returns, respectively, a void function or an error. On the other hand, 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'.

It should be noted that, as we have used a d_{max} value equal to 4 in MagicHaskeller, the obtained solutions can concatenate up to 4 functions or constants. Since we want to compare the results in similar conditions, we assume that the number of actions which can be used by Trifacta Wrangler to obtain a solution is similar to the d_{max} value in MagicHaskeller. Therefore, we limit the maximum number of actions in each Wrangler recipe to 4. Since Trifacta Wrangler uses all the elements in the column, the solution presented here tries to solve, at least, the first example.

In Table D.11 we first compare the results (in terms of 'depth') of the dynamic BK and Trifacta Wrangler. Note that, in both approaches, the actual number of functions or actions needed to solve the problem (d) can be smaller than d_{max}. We can see that Trifacta Wrangler is able to detect some data types or domains, for instance: 'url', 'time', 'phone'. With this predefined formats the tool is capable of solve very domain-specific problems such as get the name of the month or the day in a date, detect an email or extract the hour of a time. Although Trifacta Wrangler allows the user to select the type of data used and then it solves input problems according on it, there are some limitations when dealing with different formats. For instance, when using the dataset #8 from Table 4.6, the type of the data is 'dates', but each instance is written in a different

format: Trifacta Wrangler is not able to detect that they are all dates and throws an "invalid types" error. On the contrary, the dynamic background knowledge is able to use the functions adapted to this domain, regardless of their written format. Another problem here is that Trifacta Wrangler is unable to introduce new characters or constants (such as '@', 'th', ':00', etc.), the user can introduce

id	Trifacta Wrangler	d	Dynamic background knowledge	d
1	extractpatterns type: custom col: input1 on: 'digit2' extractpatterns type: custom col: input1 on: 'digit2' limit: 2 extractpatterns type: custom col: input1 on: 'digit2' limit: 3 merge col: input5,input6,input7 with: '-' as: 'column1'	4	joinArrayWithPunctuation (splitStringWithoutPunctuation a) dash	3
3	extractpatterns type: custom col: column2 on: 'dd' extractpatterns type: custom col: column2 on: 'mm' limit: 2 extractpatterns type: custom col: column2 on: 'yyyy' merge col: column1,column5,column7 with: '' as: 'column8'	4	concat (addPunctuationString a dash)	3
6	replacepatterns col: column2 with: '-' on: 'delim' global: true	1	changePunctuationString a dash	2
8	extractpatterns type: custom col: column2 on: 'dd' limit: 2	1	getDayCardinalString a	1
11	extractpatterns type: custom col: column2 on: 'dd' textformat col: column1 type: suffix text: 'th'	2	getDayOrdinal a	1
13	extractpatterns type: custom col: column2 on: 'month'	1	getMonthNameString a	1
15	extractpatterns type: custom col: column2 on: 'dayofweek'	1	getWeekDayString a	1
17		-	reduceMonth (convertMonth (getMonthString a))	3
19		-	joinArrayWithPunctuation (changeDateFormat a mdy) slash	4
21	merge col: column2,column3 with: '@' as: 'column1' textformat col: column1 type: suffix text: ',com' replacepatterns col: column1 with: '' on: ',' global: true textformat col: column1 type: removewhitespace	4	joinStringsWithAt a (append dotcom b)	3
24	extractpatterns type: custom col: column2 on: 'url' start: '@' end: '.'	1	getWordsAfterAt a	1
28	extractpatterns type: custom col: input1 on: /['@'].+[.]/ extractpatterns type: custom col: input2 on: 'alpha+'	2	getWordsBeforeDot (getWordsAfterAt a)	2
30	extractpatterns type: custom col: column2 on: 'url' start: '*' end: '@'	1	getWordsBeforeAt a	1
31		-	addNomenclature a b	1
33	extractpatterns type: custom col: column2 on: 'url' start: '*' end: '@'	1	getNomenclature a	1
35	extractpatterns type: custom col: column2 on: 'alpha2' extractpatterns type: custom col: column2 on: 'alpha2' start: 'delim' end: 'lower+' textformat col: column5,column1 type: lowercase merge col: column1,column5 as: 'column6'	4	loginByNameString a	1
37	extractpatterns type: custom col: column2 on: 'alpha+' limit: 3 extractpatterns type: custom col: column5 on: 'upper' merge col: column6,column7 with: ', ' as: 'column8' textformat col: column8 type: suffix text: '.'	4	reduceNameWithSurname a	1
46		-	addPhonePrefixByCountry a b	1
51	replacepatterns col: column2 with: '' on: '(' global: true replacepatterns col: column2 with: '' on: ')' global: true	2	deleteParentheses a	1
53	extractpatterns type: custom col: column2 on: 'phone'	1	getPhoneNumber a	1
55	merge col: input2,input1 with: '-' as: 'column1' replacepatterns col: column1 with: '-' on: ' '	2	addPhonePrefix a b	1
60	extractpatterns type: custom col: input1 on: 'digit3' extractpatterns type: custom col: input1 on: 'digit3' limit: 2 extractpatterns type: custom col: input1 on: 'digit4' start: 'digit6' end: 'end' merge col: input2,input4,input5 with: '-' as: 'column1'	4	joinArrayWithPunctuation (splitStringWithoutPunctuation a) dash	3
62		-	increaseHour a b	1
64	textformat col: column2 type: suffix text: ':00'	1	appendOclockTime a	1
67	merge col: input1,input2 with: ':' as: 'column1'	1	appendSomeTime a b	1
68		-	convertTimeTo24hoursFormat a	1
70		-	convertTimeFormat a b	1
72		-	convertTimeTo12hoursFormat a	1
74		-	convertTimeByTimeZone a b c	1
78	replacepatterns col: column2 with: '' on: ':digit2end'	1	deleteLastTimePosition a	1
80	extractpatterns type: custom col: column2 on: 'digit+'	1	getHour a	1
82	extractpatterns type: custom col: input1 on: 'digit+' limit: 2 drop col: input2 action: Drop	2	getMinutes a	1
84	extractpatterns type: custom col: column2 on: 'time'	1	getTime a	1
86		-	unitsConversion (getValue a) (getUnits a) b	3
90		-	getSystem a	1
92	extractpatterns type: custom col: column2 on: 'lower+'	1	getUnits a	1
94	extractpatterns type: custom col: column2 on: 'digit2.digit2'	1	getValue a	1

Table D.11. Functions obtained by our approach (Dynamic background knowledge) compared with 'recipe' expressions of *Trifacta Wrangler*. d is the 'depth' of the solution obtained with a $d_{max} = 4$.

them and prefixes or suffixes and this implies the need of more than four actions to deal with some examples. Furthermore, it also uses very strict predefined formats for different types of data (such as dates or times) which lead to errors when small variations in the input formats occur. On the contrary, our approach faces this sort of problems in a different way by considering constants (such as '@') as functions, although it needs a higher number of them when there are more than one input. The last and most important problem related to Trifacta is that the user needs to know the language behind the tool or some regular expressions in order to solve more complex examples. In order to overcome this problem, our system is able to solve most of the problems using only one example given by the user, without the need of having any technical knowledge.

D.3.3 Accuracy

Tables D.12, D.13, D.14, D.15, D.16, D.17 show some illustrative outcomes obtained for some datasets as well as the accuracy values for each dataset. The first instance (in italics) for each dataset (*input* column) is the one used for inferring the solution for each tool. For each dataset only the three first instances are shown.

In Table D.12 we can see the problem of having different formats in the data of one column. The datasets related with dates have very differents types of dates as examples. Our system is able to detect the different formats and deal with them. On the contrary, Flashfill and Trifacta Wrangler are incapable of detect the different types of dates to work with them or types of dates different that their predefined ones. For instance, FlashFill can not detect '11.02.96' as a date. On the other side, D.14 shows how FlashFill and Trifacta Wrangler fail when they has to deal with people's names.

In Table D.13 we observe that the three approaches work well with emails. Our approach and Trifacta Wrangler are able to detect emails. FlashFill for its part is able to work with basic string manipulation functions to deal very well with email problems.

In Tables D.15, D.16 and D.17 we can see that although FlashFill is not able to detect many types of data, is capable of solving some examples by using its DSL based on basic string manipulation problems. Here, we can see some strength and weakness in each tool. It is clear that a DSL is not enough to deal with a high range of problems when they become into a domain-related problems. In the same way, even when Trifacta Wrangler is able to solve more problems than FlashFill detecting the domains, it is important to notice that the user needs a high degree of knowledge about the problem to solve, and the language of the tool. In summary, the results show that our approach is autonomous as it recognises the domain and it is more effective in terms of results.

id	input	output	FlashFill	Trifacta Wrangler	Dynamic BK
	20040717	*17-07-2004*			
3	20021015	15-10-2002	17/10/2002		15-10-2002
	09292015	29-09-2015	17/20/0929		29-09-2015
		Accuracy:	0	0	1
	29/03/86	*29-03-86*			
6	11.02.96	11-02-96	11.02.96	11-02-96	11-02-96
	12/10/99	12-10-99	12-10-99	12-10-99	12-10-99
		Accuracy:	0.6	1	1
	03/29/86	*29*			
8	74-03-31	31	3	03	31
	25-08-85	25	8	08	25
		Accuracy:	0	0	1
	3/29/86	*29th*			
11	12/99/13	13th	99th	th	13th
	10 12 69	10th	12th	12th	10th
		Accuracy:	0	0	1
	2 of September of 2010, Monday	*September*			
13	13 November 2008	November	2008	November	November
	June 23, 2007	June	2007	June	June
		Accuracy:	0.2	1	1
	Sunday, 9 November 2014	*Sunday*			
15	2 of September of 2010, Monday	Monday	2 of September of 2010	Monday	Monday
	Wednesday, 15 October 2003	Wednesday	Wednesday	Wednesday	Wednesday
		Accuracy:	0.8	1	1
	15/02/84	*Feb*			
17	11/30/2017	Nov	Feb		Nov
	28/12/2004	Dec	Feb		Dec
		Accuracy:	0	0	1

Table D.12. Example of the results of our approach (Dynamic background knowledge) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *dates*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

id	input	output	FlashFill	Trifacta Wrangler	Dynamic BK
	Sophia & domain	*Sophia@domain.com*			
21	Logan & domain	Logan@domain.com	Logan@domain.com	Logan@domain.com	Logan@domain.com
	Lucas & domain	Lucas@domain.com	Lucas@domain.com	Lucas@domain.com	Lucas@domain.com
		Accuracy:	1	1	1
	Nancy.FreeHafer@fourthcoffee.com	*fourthcoffee.com*			
24	Andrew.Cencici@northwind-traders.com	northwind-traders.com	northwind-traders.com	northwind-traders.com	northwind-traders.com
	Jan.Kotas@litwareinc.com	litwareinc.com	litwareinc.com	litwareinc.com	litwareinc.com
		Accuracy:	1	1	1
	Nancy.FreeHafer@fourthcoffee.com	*fourthcoffee*			
28	Andrew.Cencici@northwind-traders.com	northwind-traders	northwind-traders	northwind-traders	northwind-traders
	Jan.Kotas@litwareinc.com	litwareinc	litwareinc	litwareinc	litwareinc
		Accuracy:	1	1	1
	Nancy.FreeHafer@fourthcoffee.com	*Nancy.FreeHafer*			
28	Andrew.Cencici@northwind-traders.com	Andrew.Cencici	Andrew.Cencici	Andrew.Cencici	Andrew.Cencici
	Jan.Kotas@litwareinc.com	Jan.Kotas	Jan.Kotas	Jan.Kotas	Jan.Kotas
		Accuracy:	1	1	1

Table D.13. Example of the results of our approach (Dynamic background knowledge) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *emails*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

id	input	output	FlashFill	Trifacta Wrangler	Dynamic BK
	Dr. Mark Sipser	*Dr.*			
33	Louis Johnson, PhD	PhD	Lou		PhD
	Robert Mills		Rob		
		Accuracy:	0.4	0.4	1
	Guillermo Filiepatos	*gufi*			
35	Federico A. Fithsakampf	fefi	fe	fefi	fefi
	Carmen Funcsrentano	cafu	cafuncs	cafu	cafu
		Accuracy:	0	0.6	1
	Dr. Eran Yahav	*Yahav, E.*			
37	Prof. Kathleen S. Fisher	Fisher, K.	Fisher, Kathleen S.	S, K.	Fisher, K.
	Ken McMillan, II	McMillan, K.	II, M.	II, M.	McMillan, K.
		Accuracy:	0	0.2	1

Table D.14. Example of the results of our approach (Dynamic background knowledge) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *names*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

id	input	output	FlashFill	Trifacta Wrangler	Dynamic BK
	235-7654 & Taiwan	*(886) 235-7654*			
46	17-455-81-39 & Spain	(34) 17-455-81-39	(886) 17-455-81-39		(34) 17-455-81-39
	25-437-96-20 & South Korea	(82) 25-437-96-20	(886) 25-437-96-20		(82) 25-437-96-20
		Accuracy:	0	0	1
	785-4210 & MDG	*(261) 785-4210*			
48	352-7960 & KWT	(34) 17-455-81-39	(261) 17-455-81-39		(34) 17-455-81-39
	846-2730 & AND	(376) 846-2730	(261) 846-2730		(376) 846-2730
		Accuracy:	0	0	1
	(693)-785-4210	*693-785-4210*			
51	(481)-352-7960	481-352-7960	568-734-2190	481-352-7960	481-352-7960
	(568)-734-2190	568-734-2190	568-734-2190	568-734-2190	568-734-2190
		Accuracy:	1	1	1
	425-457-2130, DJs flock . . . : 18:95	*425-457-2130*			
53	John DOE 3 Data [TS]865-000-0000 . . .	865-000-0000	John DOE 3 Data [TS]865-000-0000 . . .		865-000-0000
	17:58-19:29, 425-743-1650	425-743-1650	17:58-19:29	425-743-1650	425-743-1650
		Accuracy:	0	0.8	1
	235 7654 & 425	*425-235-7654*			
55	745-8139 & 425	425-745-8139	425-745-8139	425-745-8139	425-745-8139
	437-9620 & 425	425-437-9620	425-437-9620	425-437-9620	425-437-9620
		Accuracy:	1	1	1
	3237087700	*323-708-7700*			
60	1635879240	163-587-9240	163-587-9240	163-587-9240	163-587-9240
	1854379620	185-437-9620	185-437-9620	185-437-9620	185-437-9620
		Accuracy:	1	1	1

Table D.15. Example of the results of our approach (Dynamic background knowledge) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *phones*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

id	input	output	FlashFill	Trifacta Wrangler	Dynamic BK
	01:34:00 + 5	*06:34:00*			
62	01:55 + 5	06:55	6:55:00		06:55
	16:15:12 + 5	21:15:12	6:15:00		21:15:12
		Accuracy:	0	0	1
	01:34	*01:34:00*			
64	07:05	07:05:00	07:05:00	07:05:00	07:05:00
	16:15:12	16:15:12	16:15:12:00	16:15:12:00	16:15:12
		Accuracy:	0.8	0.8	1
	01:34 + 30	*01:34:30*			
66	01:55 + 30	01:55:30	01:55:30	16:15:12	01:55:30
	16:15:12 + 30	16:15:12	16:15:12:30	16:15:12:30	16:15:12
		Accuracy:	0.6	0.8	1
	1:34:00 PM CST	*13:34:00*			
68	3:40 AM	03:40	10:40:00	3:40	03:40
	07:05:59	07:05:59	22:05:59	07:05:59	07:05:59
		Accuracy:	0	0.6	0.8
	01:34:00	*01:34*			
78	01:55	01	0:00	01	01
	16:15:12	16:15	16:15	16:15	16:15
		Accuracy:	0.2	1	1
	01:55	*01*			
80	03:40 AM	03	03	03	03
	08:40 UTC	08	03	08	08
		Accuracy:	1	1	1
	1:34:00 PM CST	*34*			
82	3:40 AM	40	40	40	40
	07:05:59	05	05	05	05
		Accuracy:	1	1	1
	1:34:00 PM CST	*1:34:00*			
84	3:40 AM	3:40	3:40:00	3:40	3:40
	07:05:59	07:05:59	07:05:59	07:05:59	07:05:59
		Accuracy:	0.4	1	1

Table D.16. Example of the results of our approach (Dynamic background knowledge) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *times*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

id	input	output	FlashFill	Trifacta Wrangler	Dynamic BK
	1441.8mg $\rightarrow$ *g*	*1.4418001*			
86	87 s $\rightarrow$ *ns*	8700000.0	87418001		8700000.0
	1854 dam $\rightarrow$ *dm*	185400.0	18418001		185400.0
		Accuracy:	0	0	1
	56.77cl	*Volume*			
90	84kg	Mass	Volume		Mass
	1854 dam	Length	Volume		Length
		Accuracy:	0	0	1
	56.77cl	*cl*			
92	84kg	kg	kg	g	kg
	1854 dam	dam	dam	dam	dam
		Accuracy:	1	0.8	1
	56.77cl	*56.77*			
94	84kg	84	84		84
	1854 dam	1854	1854		1854
		Accuracy:	1	0.4	1

Table D.17. Example of the results of our approach (Dynamic background knowledge) compared with *FlashFill* and *Trifacta Wrangler* using datasets of *units*. *Output* is the expected output. The first row of each dataset is the example given to *FlashFill* and *MagicHaskeller* to learn, and used in Wrangler to generate the results. Green colour means correct result; Red colour means incorrect result.

D.4 Tool: System Overview

This description of the system has been published in [22, 23].

BK-ADAPT uses MagicHaskeller [90] as the underlying IP system. BK-ADAPT works as follows: (1) one output example is filled (See Figure D.1) and used to detect the appropriate domain (See Figure D.2) and set of primitives that form the dynamic background knowledge ; (2) using the background knowledge and the example, MagicHaskeller learns a function `f` that correctly transforms the input of the example to the given output (See Figure D.3); and (3) the function `f` is applied to the rest of the inputs, obtaining the new values for the output column automatically (See Figure D.4).

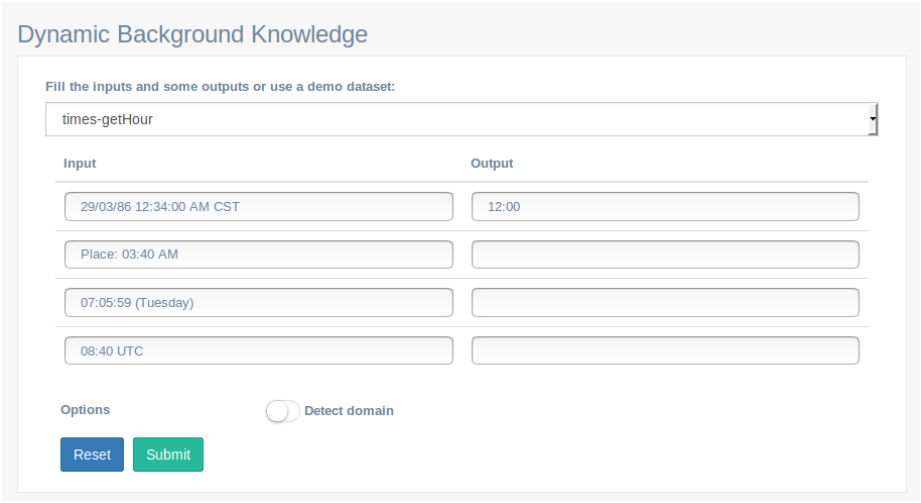


Figure D.1. Interface of BK-ADAPTwith a form where the user can select a dataset and fill one or more outputs.

The background knowledge is composed of *basic string manipulation functions* and functions that can be useful for the specific problems of each domain as well. The list of domains for this demo is: dates, emails, names, phones, times and units. However, this list can be increased and/or include different domains. More concretely, the background knowledge to be used for MagicHaskeller to solve one given example is dynamically generated in two different ways: (1 *Inferred Domain*) The domain is automatically detected and the correct background knowledge is used; (2 *Dynamic background knowledge by primitive ranking*). The background knowledge is formed by those primitives that are more likely to be needed for solving the example. To this end, the BK-ADAPT system uses two learning modules trained using the descriptive meta-features previously extracted from the examples: a domain classifier, that predicts the domain an example belongs to, and a *primitive estimator* that estimates the probability of each primitive to be used for solving the example. The background knowledge is

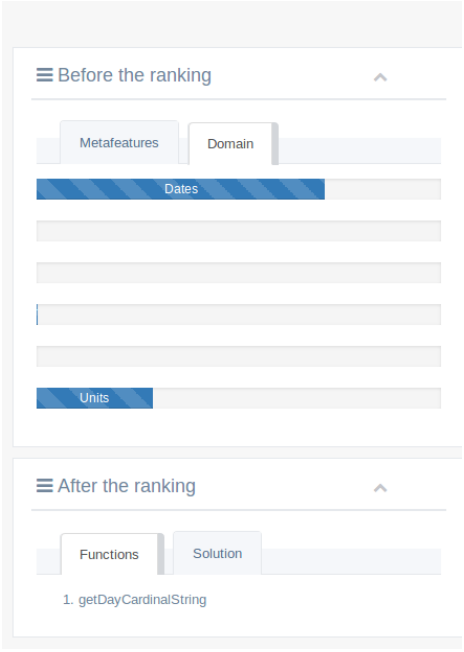


Figure D.2. BK-ADAPTis able to recognise the domain and it is shown to the user.

dynamically generated depending on the problem to solve[1] [2].

D.4.1 System Architecture and User Interface

BK-ADAPT has been developed as a user-friendly PHP-based web application (see Figure D.5) which main interface simulates a spreadsheet or table with input/output text fields. In this interface, the input field is used as a way to provide the original value for the attribute we want to transform, and the output field is the result the user wants to obtain. The goal of the system is, given just one (or very few) input/output example(s), try to fill the outputs of the rest of instances whose output fields have not been filled. For this, the user can fill as many output text field as they like (from 1 to $n-1$) in order to show the system how the output data should look like, (i.e., which is the desired data transformation). The transformation process is completely automatic and transparent for the user: no need to reload the website as the system works dynamically through AJAX requests (jQuery framework). [3]

[1] The complete description of the approach can be found in section X and [21]
[2] The code is available at: https://github.com/liconoc/DataWrangling-DSI
[3] The functionality of our system can be seen on: https://www.youtube.com/watch?v=wxFhXYyonOw

Figure D.3. The system shows to the user the list of meta-features extracted from the examples and the background knowledge generated.

As we have seen, data wrangling is widely reported as an intense manual process for data scientists which manual involvement affects the time needed to finish a project or data analysis. BK-ADAPT allows the user to use the time in other and more important tasks by three reasons: (1) do not do things by hand; (2) do not write code; and (3) use a tool that does the task automatically.

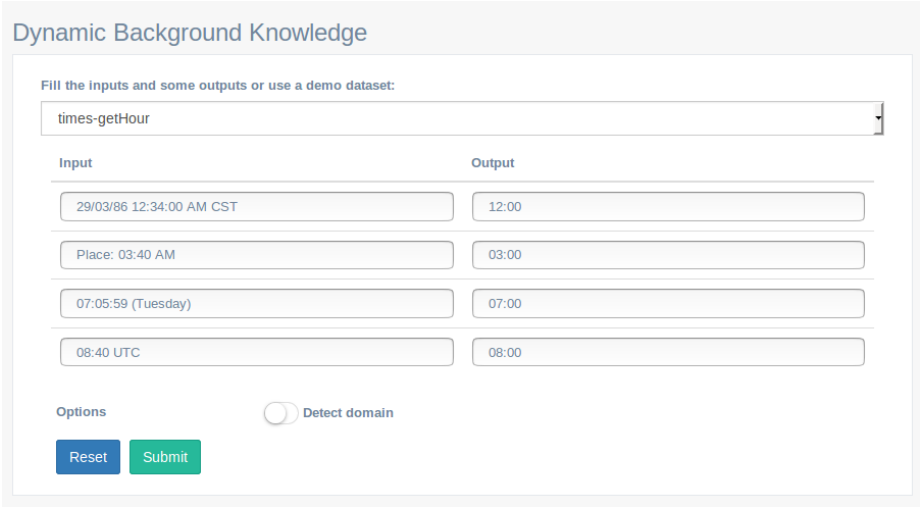


Figure D.4. Once the process is ended, BK-ADAPT fills automatically the rest of the output.

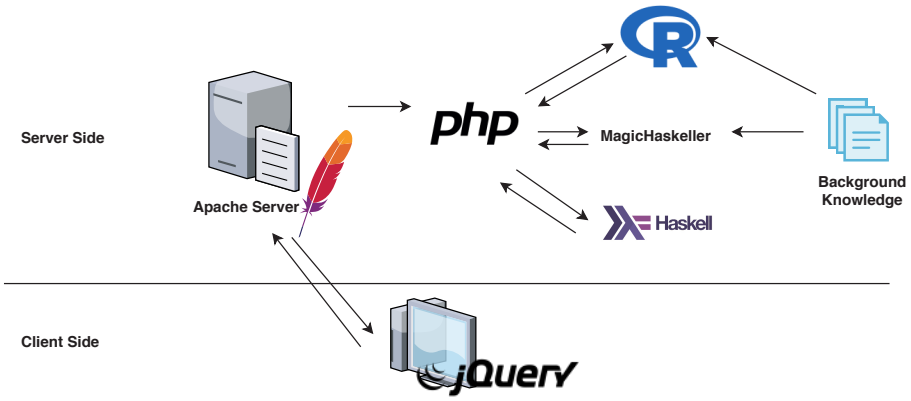


Figure D.5. : system architecture.

Appendix E

AUTOMAT[R]IX : Supplementary Material

E.1 Background Knowledge: List of Functions

id	f	m_i	n_i	m_o	n_o	m_min	n_min
01	rowSums(A)	m	n	1	m	2	NA
02	colSums(A)	m	n	1	n	2	NA
03	rowMeans(A)	m	n	m	1	NA	NA
04	colMeans(A)	m	n	1	n	2	NA
05	nrow(A)	m	n	1	1	2	NA
06	ncol(A)	m	n	1	1	2	NA
07	cor(A)	m	n	n	n	2	NA
08	det(A)	m	m	1	1	2	NA
09	diag(A)	m	n	1	n	2	NA
10	t(A)	m	n	n	m	NA	NA
11	is.na(A)	m	n	m	n	NA	NA
12	!is.na(A)	m	n	m	n	NA	NA
13	A==0	m	n	m	n	NA	NA
14	A!=0	m	n	m	n	NA	NA
15	A<=0	m	n	m	n	NA	NA
16	A>=0	m	n	m	n	NA	NA
17	A<0	m	n	m	n	NA	NA
18	A>0	m	n	m	n	NA	NA
19	A==1	m	n	m	n	NA	NA
20	A!=1	m	n	m	n	NA	NA
21	A<=1	m	n	m	n	NA	NA
22	A>=1	m	n	m	n	NA	NA
23	A<1	m	n	m	n	NA	NA
24	A>1	m	n	m	n	NA	NA
25	A*A	m	n	m	n	NA	NA
26	apply(A,1,rev)	m	n	m	n	2	NA
27	apply(A,2,rev)	m	n	m	n	2	NA
28	as.table(A)	m	n	m	m	NA	NA
29	cbind(A,A)	m	n	m	n*2	NA	NA
29	rbind(A,A)	m	n	m*2	n	NA	NA
30	length(A)	m	n	1	1	NA	NA
31	max(A)	m	n	1	1	NA	NA

continued on next page

continued from previous page

id	f	m_i	n_i	m_o	n_o	m_min	n_min
32	min(A)	m	n	1	1	NA	NA
33	mean(A)	m	n	1	1	NA	NA

Table E.1. List of functions from the R language included in the background knowledge of AUTOMAT[R]IX. $m_i x n_i$ represents the size of the input, while $m_o x n_o$ represents the size of the output once the function is applied. m_{min} and n_{min} are the minimum number of rows and columns that the input matrix needs to accomplish.

E.2 Prior Probabilities: List of Top Functions in GitHub

library	content	count
base	length	634503
base	nrow	171765
base	is.na	159483
base	which	154866
base	colnames	151345
base	max	107362
base	unique	92035
base	mean	89843
base	cbind	88746
base	dim	85965
base	ncol	83110
base	t	76742
base	min	72586
base	rbind	71891
base	class	63129
base	order	44862
base	sort	31890
base	diag	26276
base	rowSums	15053
base	colSums	11226
base	sin	10286
base	duplicated	9914
base	solve	9693
base	isTRUE	9314
base	colMeans	8740
base	cos	8366
base	rowMeans	7443
base	which.max	6743
base	which.min	5556
base	det	1783
base	upper.tri	1739
Matrix	solve	988
base	as.table	725

Table E.2. List of the R functions most used in GitHub. This table only contains those functions related to matrices. It includes the library where the function is included and the absolute number of uses (count).

E.3 Text Hints: Frequent Terms from Function's Documentation

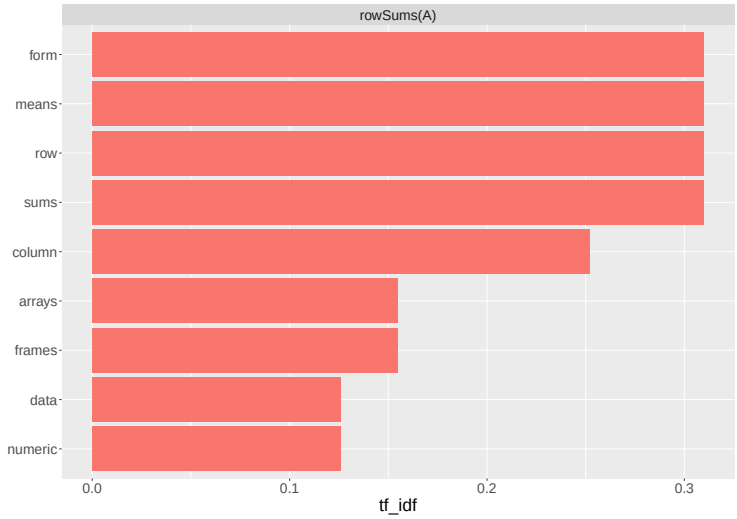


Figure E.1. TF-IDF values for the R primitives `rowSums`, `colSums`, `rowMeans` and `colMeans` extracted from the R help documentation.

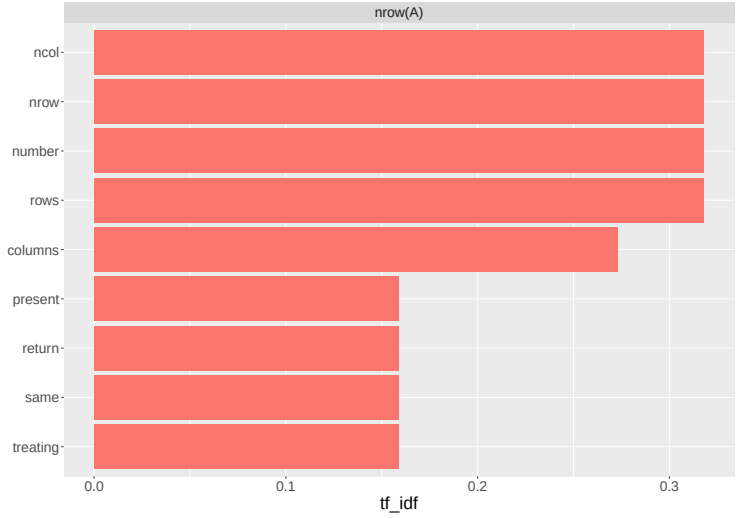


Figure E.2. TF-IDF values for the R primitives `nrow` and `ncol` extracted from the R help documentation.

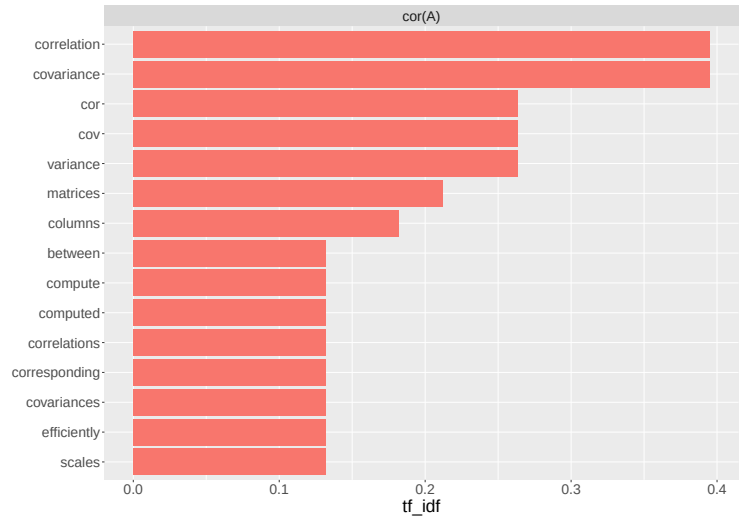


Figure E.3. TF-IDF values for the R primitive `cor` extracted from the R help documentation.

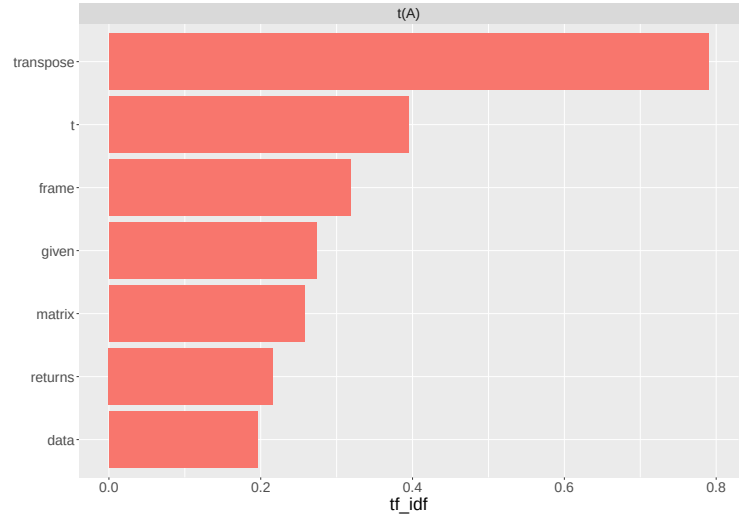


Figure E.4. TF-IDF values for the R primitive `t` extracted from the R help documentation.

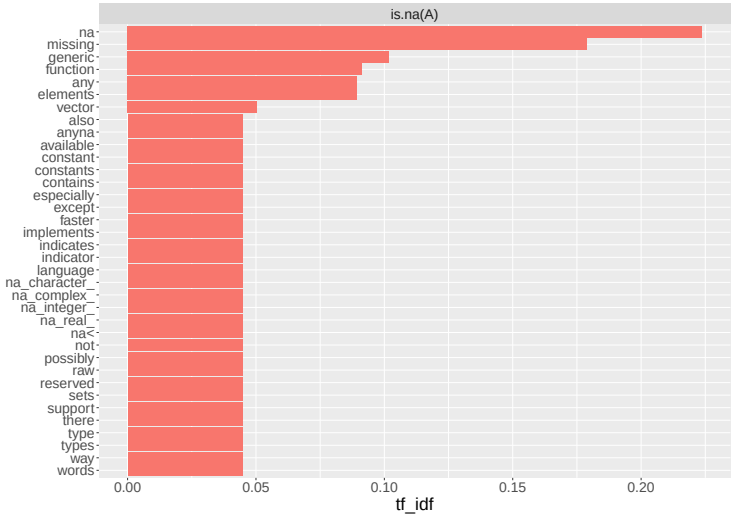


Figure E.5. TF-IDF values for the R primitive `is.na` and `!is.na` extracted from the R help documentation.

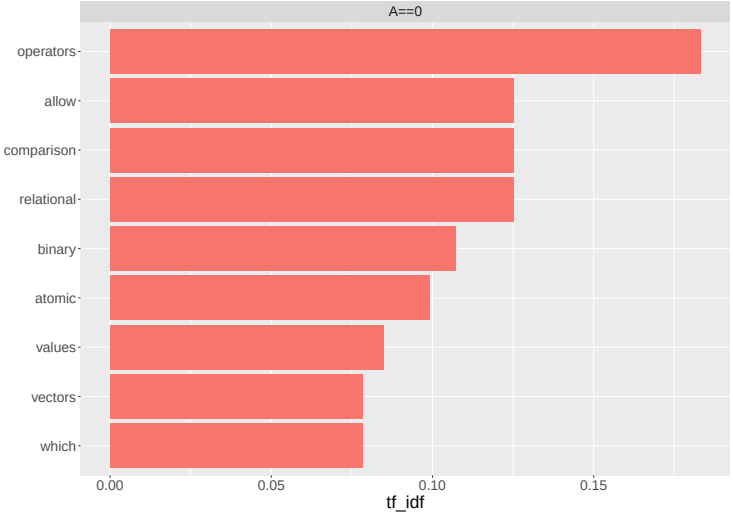


Figure E.6. TF-IDF values for the R primitives related to operators, like `A==0`, extracted from the R help documentation.

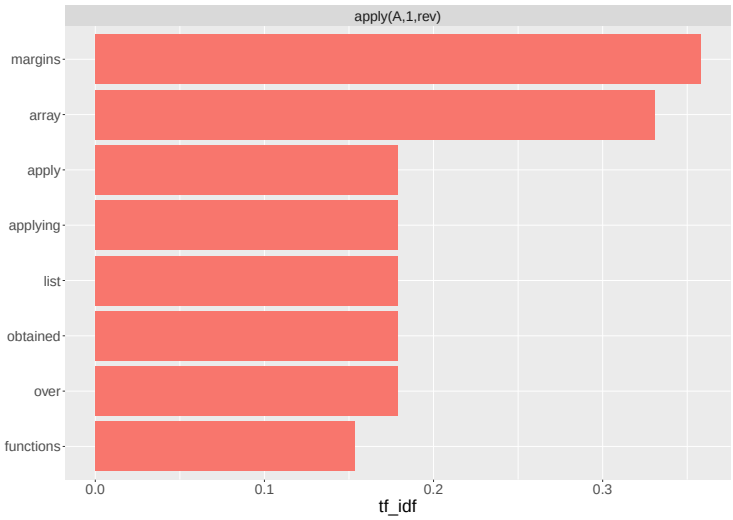


Figure E.7. TF-IDF values for the R primitives related to `Apply` extracted from the R help documentation.

E.4 Data: List of Examples

example	text
01	How to reverse a matrix
02	replace 0's with 1's
03	Get positions for NA
04	rowsums accross specific row in a matrix
05	How to get the sum of each four rows of a matrix in R
06	Extract diagonal of a matrix in R
07	Rotate a Matrix in R
08	Concatenating matrices by row in r
09	Concatenating matrices by column in r
10	Convert a matrix in r into a upper triangular matrix
11	Multiply a matrix by another matrix
12	How can I create a correlation matrix in R?
13	which elements are greather than one
14	which elements are negative numbers
15	What is the best way to transpose a matrix in R
16	How to get the mean over a entire matrix
17	How to get the max over a entire matrix
18	How to get the min over a entire matrix
19	Elegant way to report the number of missing values in a matrix
20	Na's in a diagonal matrix

Table E.3. Description of the problems (questions from users) used for learning matrix transformations. The text is used as a hint in natural language provided by the user.

E.5 Pipeline of events for a simple example

NA	1	NA	1
1	NA	NA	2
NA	2	NA	3

(a) Matrix A with some NA values.

TRUE	FALSE	·	·
·		·	·
·		·	·

(b) Matrix B with Boolean values representing whether a cell contains a NA or not. Only two cells are completed.

TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	TRUE	FALSE

(c) Matrix S, the completed version of B.

Iteration: 1		
background knowledge (ordered by the probabilities):		
1	**length(A)**	**0.239125**
2	!is.na(A)	0.1178071
3	is.na(A)	0.1178071
4	apply(A,1,rev)	0.05579061
5	apply(A,2,rev)	0.05579061
Variables:		
	$g \leftarrow$ **length(A)**	
	$found \leftarrow$ FALSE	
	$explored \leftarrow$ FALSE	
	Number of explored nodes: 1	
	Number of created nodes: 68	
Iteration: 2		
background knowledge (ordered by the probabilities):		
1	**!is.na(A)**	**0.1178071**
2	is.na(A)	0.1178071
3	length(length(A))	0.05718077
4	apply(A,1,rev)	0.05579061
5	apply(A,2,rev)	0.05579061
Variables:		
	$g \leftarrow$ **!is.na(A)**	
	$found \leftarrow$ FALSE	
	$explored \leftarrow$ FALSE	
	Number of explored nodes: 2	
	Number of created nodes: 34	
Iteration: 3		
background knowledge (ordered by the probabilities):		
1	**is.na(A)**	**0.1178071**
2	length(length(A))	0.05718077
3	apply(A,1,rev)	0.05579061
4	apply(A,2,rev)	0.05579061
5	nrow(A)	0.05569636
Variables:		
	$g \leftarrow$ **is.na(A)**	
	$found \leftarrow$ TRUE	
	$explored \leftarrow$ FALSE	
	Number of explored nodes: 3	
	Number of created nodes: 0	

Time: 0.033s

Figure E.8. Short example of the pipeline of events for a simple problem: marking the positions with NA.

Figure E.8 shows an example of a pipeline of events when running the algorithm. In this example, the goal is to learn a function that given an input matrix, returns an output matrix where each cell contains a Boolean value indicating if the corresponding position in the input matrix is `NA`. In the top of the figure, we include: matrix A (input), matrix B (partial output) and matrix S (complete output). The text used as a hint is: *"Get positions for NA"*. In the figure we show the three iterations of the algorithm. BK represents the five functions with highest probability for each iteration. *Variables* shows the function selected, whether the solution has been found or the three has been completely explored, the total number of nodes explored at the moment and the number of nodes created in this iteration. Note that in *Iteration 2* we explore the branch of the tree that falls under `length(A)`. One of the compositions in that branch is precisely `length(length(A))`, i.e., apply the function again, which, because the a priori probability of length alone is so high, still gets sufficient probability to appear in the top 5 in the figure.