



THE UNIVERSITY *of* EDINBURGH

This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.

A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.

This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Probabilistic Type Inference for the Construction of Data Dictionaries

Taha Yusuf Ceritli

Doctor of Philosophy
Institute for Adaptive and Neural Computation
School of Informatics
The University of Edinburgh
2021

Abstract

The data understanding stage plays a central role in the entire process of data analytics, as it allows the analyst to gain familiarity with the data, identify data quality issues, and discover initial insights into the data before further analysis (Chapman et al., 2000). These tasks become easier in the presence of well-documented background information such as a data dictionary, which is defined as “a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format” (McDaniel, 1994). However, data dictionaries are often missing or incomplete.

In this thesis we focus on inference of data types (both syntactic and semantic), and develop probabilistic approaches that enable the automatic construction of a data dictionary for a given dataset. Unlike existing rule-based methods, our proposed methods allow us to express uncertainty in a principled way and can provide accurate type predictions even for messy datasets with missing and anomalous values.

The thesis makes the following contributions: First, we present `p`type - a probabilistic generative model that uses Probabilistic Finite-State Machines (PFSMs) to represent data types. By detecting missing and anomalous data, `p`type infers syntactic data types accurately and improves over the performance of existing approaches for type inference. Moreover, it offers the advantage of generating weighted predictions when a column of messy data is consistent with more than one type assignment, in contrast to more familiar finite-state machines (e.g., regular expressions).

Secondly, we propose `p`type-cat which is an extension of `p`type for a better detection of the categorical type. `p`type treats non-Boolean categorical variables as either integers or strings. By combining the output of `p`type and additional features that can indicate whether a column represents a categorical variable or not, `p`type-cat can correctly detect the general categorical type (including non-Boolean variables). In addition, we adapt `p`type to the task of identifying the values associated with the corresponding categorical variable.

Finally, we present `p`type-semantics to demonstrate how `p`type can be enriched by semantic information. In this regard, we focus on *dimension* and *unit* inference, which are respectively the task of identifying the *dimension* of a data column and the task of identifying the *units* of its entries. Syntactic type inference methods including `p`type do not address these tasks. However, `p`type-semantic can extract extra semantic information (such as dimension and unit) about data columns and treat them as either floats or integers rather than strings.

Lay Summary

A fundamental problem in data analytics is to understand the basic properties of a given dataset. Storing such information in a separate document such as a data dictionary, which is defined by McDaniel (1994) as “a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format”, can help to re-use a dataset. Although these documents are often missing or incomplete in practice, they can be automatically created based on the data itself.

In this thesis, we focus on predicting the data type (such as categorical, date, float, integer and string) for each column of a dataset. Relying on both the syntax and the semantics, our models can provide accurate type predictions even in the presence of missing and anomalous data. Note that missing data refers to the absence of a value in a data entry which can be represented by special codes such as `NA` and `null`, while anomalous data refers to the presence of an inconsistent data value such as `ERROR` in a column of integers.

We first present `ptype` - a probabilistic model of a data column which potentially consists of missing and anomalous data entries. We represent each data type in a probabilistic manner and propose a model that can provide weighted predictions for a column of data consistent with more than one type, in contrast to non-probabilistic approaches. Our experiments show that `ptype` improves over the performance of existing type inference methods.

A limitation of `ptype` is that it treats categorical variables with more than two possible values as either integers or strings (e.g., `ptype` would treat the blood type of a person encoded by `A`, `B`, `AB` or `O` as string rather than categorical). We address this limitation by combining its output with additional features that can indicate whether a column represents a categorical variable or not, and running a separate classifier to distinguish between the categorical type and the integer/string types. In addition, we adapt `ptype` to the task of identifying the values associated with the corresponding categorical variable.

The two methods described above rely on the syntax of the entries of a data column for type inference; however, they do not consider the semantic information about a data column. We demonstrate how `ptype` can be enriched semantically by incorporating information from external sources such as a knowledge graph. In particular, we tackle *dimension* and *unit* inference, which are respectively the task of identifying the *dimension* (e.g., length and volume) of a data column and the task of identifying the *units* (e.g., metres and litres) of its entries.

Acknowledgements

I would not be able to complete this dissertation without the support of many colleagues and friends.

First of all I would like to thank my supervisor, Chris Williams, who has been a source of inspiration and guidance for me. I feel privileged to be supported with his wisdom and dedication in the research, which did not only stimulate the progress in this thesis but also will stay as an inspiration for me going forward as a researcher. I am also grateful to James Geddes for his collaboration and support throughout my studies. It has always been a pleasure to receive feedback from James. I would also like to thank Charles Sutton and Ewan Klein for providing valuable feedback during my yearly review meetings, which helped to shape this thesis. Finally, I thank Roly Perera for his collaboration on software development, which did not only enhanced the quality of my existing code but also improved my understanding.

I owe the Alan Turing Institute a debt of gratitude for funding my studies and hosting me. I am also grateful to the School of Informatics at the University of Edinburgh for supporting me with a stimulating research environment.

Thanks also to Alfredo, Deniz, Julien and Gerrit for their friendships; Gülce, Nadin, Nezihe, Selin and many more, who have created wonderful memories over the past few years. Finally, I would like to express my dearest gritudes to my family for their endless support throughout my entire life. I am deeply thankful for their unconditional love.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Taha Yusuf Ceritli)

To my grandmother Gülen

Table of Contents

1	Introduction	1
1.1	Outline of the thesis	3
2	Background	5
2.1	Data Dictionary	5
2.2	Syntactic Type Inference	7
2.3	Semantic Type Inference	11
3	Syntactic Type Inference	13
3.1	Introduction	13
3.2	Methodology	14
3.2.1	Probabilistic Finite-State Machines	14
3.2.2	The Proposed Model	17
3.2.3	Inference	18
3.2.4	Training of the Model	19
3.3	Experiments	21
3.3.1	Experimental Setup	21
3.3.2	Quantitative Results	23
3.3.3	Qualitative Results	29
3.4	Summary	34
4	Inferring the Type and Values of Categorical Variables	35
4.1	Introduction	35
4.2	Methodology	37
4.2.1	Probabilistic Type Inference for Categorical Variables	38
4.2.2	Identification of Categorical Values	39
4.2.3	Using Meta-data for Inference	39
4.3	Experiments	41

4.3.1	Experimental Setup	41
4.3.2	Experimental Results	46
4.4	Discussion	55
5	Bringing Semantics into Type Inference	56
5.1	Introduction	56
5.2	Methodology	58
5.2.1	Representing Units	59
5.2.2	The Proposed Model	60
5.2.3	Inference	62
5.3	Related Work	64
5.3.1	Semantic Web Technologies	65
5.3.2	Regular Expressions	66
5.3.3	Machine Learning	66
5.4	Experiments	68
5.4.1	Experimental Setup	68
5.4.2	Results	70
5.5	Discussion	77
6	Conclusions and Future Work	79
6.1	Summary of the Contributions	79
6.2	Future Work	81
6.2.1	Identifying the Ordinal and Nominal Types	81
6.2.2	Using Header Information to Aid Inference	81
6.2.3	Using Numeric Values for Unit Inference	82
6.2.4	Enhancing User Interactions	83
A	Appendix for ptype	85
A.1	PFSMs for Data Types	85
A.1.1	Integers	85
A.1.2	Floats	85
A.1.3	Strings	86
A.1.4	Booleans	86
A.1.5	Dates	86
A.1.6	Missing	86
A.1.7	Anomaly	87

A.2	Data Sets	87
A.3	Derivations for the Training	92
A.3.1	Derivative of L_c	93
A.3.2	Derivative of L_f	95
A.4	The Outputs of the PADS Library	96
A.5	Scalability of the Methods	99
B	Appendix for ptype-cat	100
C	Appendix for ptype-semantic	109
C.1	Additional Information about Our Knowledge Graph	109
C.2	Brief Description of the Datasets	109
C.3	Regular Expressions for Parsing Unit Symbols	110
C.4	Derivations for Inference	111
C.4.1	Column Type	111
C.4.2	Row Label	111
C.4.3	Row Unit	112
C.4.4	Column Unit	112
	Bibliography	113

List of Figures

2.1	Normal, missing, and anomalous values are denoted by green, yellow, and red, respectively.	7
3.1	Graphical representation of a PFSM with states $\theta = \{q_0, q_1, q_2\}$ and alphabet $\Sigma = \{+, -, 0, 1, \dots, 9\}$ where p denotes $\frac{1-P_{stop}}{10}$	16
3.2	Fraction of the non-type entries in a dataset, calculated by aggregating over its columns. Note that ‘overall’ denotes the fraction after aggregating over the datasets.	23
3.3	Normalized confusion matrices for (a) F#, (b) messytables, (c) ptype, (d) reader, (e) Trifacta, and (f) hypoparsr plotted as Hinton diagrams, where the area of a square is proportional to the magnitude of the entry.	25
3.4	The time in seconds taken to infer a column type with ptype, as a function of the number of unique data entries U in the column. Also shown is the line $c_0 + c_1U$, where c_0 is a small constant.	27
3.5	AUC(ptype) - AUC(Trifacta) plotted for each test dataset.	28
3.6	ROC curves plotted for some of the test datasets.	29
3.7	Annotating a subset of a T2D data set using ptype as normal, missing, and anomalous entries, denoted by green, yellow, and red, respectively.	30
4.1	Overview figure.	36
4.2	A dataset and a section of the corresponding ARFF file.	37
4.3	A graphical representation of the re-distribution step where we split the probability mass for the integer type between the integer and categorical types.	39
4.4	PR curves for the methods.	46
4.5	Hinton plots of the normalized confusion matrices.	47
4.6	PR curves for the methods.	51
4.7	Hinton plots of the normalized confusion matrices.	52

4.8	PR curves for the methods.	54
5.1	A motivating real-world example that represents our pipeline. a) shows the samples of a raw dataset. b) indicates the intermediate steps required to transform the column. c) denotes the final data column obtained by applying the transformations.	57
5.2	Normalized confusion matrices for (a) Quantulum, (b) S-NER and (c) ptype-semantics plotted as Hinton diagrams, where the area of a square is proportional to the magnitude of the entry.	72
5.3	Runtime violin plots denote the time in seconds taken to infer dimensions per column. The dot, box, and whiskers respectively denote the median, interquartile range, and 95% confidence interval.	73
6.1	The probability density functions of the numeric values	83
A.1	A fragment of the PADS output for a given dataset.	98

List of Tables

2.1	A summary of the rule-based type inference methods. Here, the presence of a ✓ sign indicates whether a method considers the corresponding issue.	8
3.1	Histogram of the column types observed in the training and the test data sets.	22
3.2	Performance of the methods using the Jaccard index and overall accuracy, for the types Boolean, Date, Float, Integer and String.	24
3.3	Performance of the methods using the Jaccard index and overall accuracy, for the types Date, Logical, Numeric and Text.	26
3.4	The percentages of FPs, FNs, and TPs for Trifacta and ptype on type/non-type detection.	29
4.1	Data types and their synonyms found in our data dictionaries.	41
4.2	Performance of the methods using the overall accuracy and per-class Jaccard index, for the Categorical, Date, Float, Integer and String types.	46
4.3	Performance of the models on inference of categorical values.	49
4.4	Performance of the methods using the overall accuracy and per-class Jaccard index, for the Categorical, Date, Float, Integer and String types. We highlight the best score in each row by making the highest score bold.	51
4.5	Performance of the methods using Overall Accuracy and average of Jaccard index per column.	54
5.1	Two elements of the unit dictionary. Note that URIs begin with <code>https://en.wikipedia.org/</code>	59
5.2	A summary of the notation used by ptype-semantics.	60

5.3	Size of the datasets used and the number of units and unit symbols in each data column.	69
5.4	Performance of the methods for dimension inference using the Jaccard index and overall accuracy, for the dimensions Currency, Data storage, Length, Mass and Volume. We highlight the best score in each row by making the highest score bold.	71
5.5	Accuracy of the methods on unit identification. We highlight the best score in each row by making the highest score bold.	74
5.6	The t-statistics and p-values obtained by applying a paired t-test on the differences of the accuracies, i.e., Accuracy(ptype-semantics) - Accuracy(competitor method).	75
5.7	Accuracies of the methods on handling unit canonicalization problems per data column.	77
A.1	Information about the data sets used.	91
A.2	A sample test dataset.	97
A.3	Size of the test datasets and the times in seconds it takes to infer column types per column (on average), where U denotes the number of unique data entries in a dataset.	99
B.1	Summaries per fold.	107
B.2	Summaries per fold.	108

Chapter 1

Introduction

Data analytics refers to the process of generating useful insights from raw datasets. The Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology formalizes this process and describes the stages that the analyst typically goes through in a data analytics project, such as “data collection”, “data understanding”, “data cleaning” and “data modeling” (Chapman et al., 2000). The phase to which the machine learning community draws attention the most is perhaps the data modeling stage where the goal is to develop a statistical model to address an analytical task. These efforts are crucial to obtain a good solution to the problem of interest; however, the modeling step is only a small part of the whole process. In practice, most of the efforts is rather spent on the concept of data wrangling, which refers to the task of understanding, interpreting and preparing a raw data set, and turning it into a usable format (see Nazábal et al. (2020) for a detailed analysis of the challenges in a data analytics project). Going back to the categorization proposed by Chapman et al. (2000), data wrangling is mainly carried out during the “data understanding” and “data cleaning” stages. It often leads to a frustrating and time-consuming process for large data sets, and even possibly for some small-sized ones (Dasu and Johnson, 2003).

In the data understanding stage, the aim is to gain familiarity with the data, identify data quality issues, and discover initial insights into the data before further analysis (Chapman et al., 2000). Therefore, it plays a central role in the entire process of data analytics. These tasks become challenging when well-documented prior information such as a data dictionary, which is defined as “a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format” (McDaniel, 1994), is not available. Given the abundance of datasets without data dictionaries, a common but ad-hoc solution to this problem is to manually inspect a given

dataset to understand its basic properties, such as what the dataset is about, what each data column represents and how missing data is encoded. The need to apply such manual efforts is perhaps one of the reasons why so much time is spent on data wrangling.

To accelerate “data understanding”, we can make use of data-driven approaches that can detect the basic properties of a given dataset based on the data itself, e.g., identifying syntactic types of the data columns such as integer and date (Petricek et al., 2016; Döhmen et al., 2017; Lindenberg, 2017; Fisher and Gruber, 2005; Wickham et al., 2017; Stochastic Solutions, 2018; Trifacta, 2018), identifying semantic entity types of the data columns such as country (Chen et al., 2019b) and identifying missing data encodings (Qahtan et al., 2018). Note that there is no consensus on the content and format of data dictionaries; however, a core component commonly reported in data dictionaries is the *data type* which specifies the type of values in a data column. The data type (syntactic and semantic) is one of the fundamental properties of a data column that needs to be understood before further analysis, including the data modeling stage.

Existing approaches provide a limited capability for type inference, which, in turn, slows down the data understanding stage. There are three main reasons for these limitations (see Chapter 2 for a detailed discussion.). First, most methods are rule-based, taking into account heuristics to identify data types. Therefore, their capabilities are restricted to the cases covered by these heuristics. Second, existing methods are not robust against missing and anomalous data, which is commonly found in raw datasets. Consequently, they fail to identify the data types in the presence of such data. Third, most previous works do not comprehensively consider the categorical type. While most methods can annotate the Boolean variables with the categorical type, they do not take into account the non-Boolean variables where there are more than two possible values encoded by integers or strings, e.g., the Blood type of people where the possible values are A, B, AB and O. As a result, the analyst needs to inspect the data to identify the data types and then perform the necessary corrections on the data type predictions obtained by using existing methods.

In this thesis, I consider three type inference problems and investigate systematic approaches to these problems. The goal here is to develop probabilistic methods that can perform better in the presence of missing and anomalous data than existing approaches, most of which are rule-based. The main contributions of this thesis can be summarized as follows:

Syntactic type inference: Syntactic type inference is typically carried out by using Finite-State Machines (FSMs), such as regular expressions, that either accept or reject

a given data value (Petricek et al., 2016; Döhmen et al., 2017; Lindenberg, 2017; Fisher and Gruber, 2005; Wickham et al., 2017; Stochastic Solutions, 2018; Trifacta, 2018). In contrast, we use Probabilistic Finite-State Machines (PFSMs) that assign probabilities to different values and therefore offer the advantage of generating weighted predictions when a column of messy data is consistent with more than one type assignment. Moreover, PFSMs provide a more adaptive solution than FSMs as they can be trained with data.

Inferring the type and values of categorical variables: Existing methods annotate non-Boolean categorical variables either as integer or string rather than categorical. To eliminate the need to manually transform such data into categorical, OpenML¹, Weka² and Majoor and Vanschoren (2018) use heuristics (see Chapter 2 for a detailed discussion). Here, I present ptype-cat - a probabilistic model that can identify the general categorical type (including non-Boolean variables) and the corresponding categorical values better than the existing solutions.

Semantic type inference: In addition to the syntactic types described above, the semantic types such as country can provide further insights into the data. The analyst can extract such information by employing entity type inference methods; however, the existing approaches do not tackle the problem of understanding how measurements are encoded in quantitative data columns. Here, I address the task of identifying the dimension (e.g., length, mass and volume) of a data column and the units of all its entries so that they all are expressed in the same units.

1.1 Outline of the thesis

The remainder of this document is structured as follows:

Chapter 2 introduces three type inference problems and presents an overview of existing methods that address these problems.

Chapter 3 describes the probabilistic model ptype that has been built for the syntactic type inference task. We propose a generative model that uses PFSMs to represent data types. The proposed model can be used to identify the data type for each column of a dataset robustly in the presence of missing and anomalous values. Our experiments demonstrate the benefits of our model in handling messy raw datasets over the

¹The code is available at <https://github.com/openml/ARFF-tools/blob/master/csv-to-arff.py> [Accessed on 09/11/2020]

²The details are at https://waikato.github.io/weka-wiki/formats_and_processing/convertting_csv_to_arff/ [Accessed on 05/12/2020].

existing approaches. The work presented in this chapter has been published in Ceritli et al. (2020).

Chapter 4 gives the details of ptype-cat which extends ptype to identify the general categorical data type, including non-Boolean variables. Besides, we identify the possible values of each categorical variable by using ptype. Combining these methods, we automate the construction of data dictionaries in the well-known Attribute Relation File Format and provide enhanced type inference capability for Pandas DataFrames. Our extensive experiments show that our method achieves better results than existing applicable solutions.

Chapter 5 demonstrates how semantic type inference can be carried out by combining knowledge graphs and probabilistic models. Particularly, we address the task of explaining how measurements are encoded in quantitative data columns. We evaluate the proposed model on several real-world datasets and show that its performance is better than the applicable related works.

Chapter 6 summarises the results presented in the thesis and discusses directions for future work.

Chapter 2

Background

In this chapter, we describe the concept of a data dictionary (Sec. 2.1) and introduce three problems: (i) syntactic type inference, (ii) identification of categorical values (both in Sec. 2.2) and (iii) semantic type inference (Sec. 2.3). Addressing these tasks can assist in constructing the data dictionary for a given dataset by providing data type-related information about data columns. In addition, we discuss existing methods that can help to automate each of these tasks.

2.1 Data Dictionary

A data dictionary (a.k.a. *metadata* or *schema*) is typically used to provide background information about a dataset, such as the data type of each column, data values used to encode missing data and data values that should be treated as anomalous. However, data dictionaries are often incomplete or incorrect as constructing a data dictionary for a given dataset can be a tedious task, and the process is prone to human errors. Below, we review the following three components of data dictionaries: data type, missing data and anomalous data. Note that Castelijns et al. (2020) use these three components to evaluate and describe the data quality in a formal framework.

Data Type

Each column of a dataset can be ascribed to a data type. These types can be basic data types (such as Boolean, date, float, integer and string) or more complex data types (such as the general categorical type, IP-address, email-address and phone number). Identifying the data types can help the analyst to decide how the data is going to be

processed by providing insight into the data. For example, the analyst may want to apply one-hot encoding on categorical variables as a data preprocessing step. Similarly, different likelihood models may be chosen for continuous and categorical variables (see Sec. 2.2 for a detailed discussion). Therefore, data type is one of the core components of data dictionaries. However, accurate type inference in the presence of missing and anomalous data remains a challenge.

Missing Data

Missing data is a significant part of data dictionaries as it can inform the analysts how the absence of data values is represented. Identifying missing data can become difficult without the help of data dictionaries as there are many ways to encode missing data. Perhaps the most natural option is to leave missing data entries empty, although it is a common practice to use special data values (e.g., NA) known as missing data encodings to represent missing data. It becomes even more challenging to identify missing data when non-standard missing data encodings are used. In such cases, we can rely on data types to detect missing data, e.g., checking whether the data type of each data value is the same as the data type of the whole column. However, relying on data types would not help when missing data is encoded by the same type as the column type (e.g., -1 or -99 in a column of the integer type). Pearson (2006) refers to such problems as the problem of *disguised missing data*, and further shows how interpreting missing data as valid values can mislead the statistical analysis. Density-based outlier detection methods can be used to detect disguised missing data (Qahtan et al., 2018).

Anomalous Data

Poor data quality does not only result from the absence of data but also from the presence of anomalous data. Chandola et al. (2009) define an anomaly as “a pattern that does not conform to expected normal behavior”. In our context, anomalies refer to unexpected or invalid entries for a given column, which might be a consequence of the data collection procedure, e.g. error and warning messages generated by servers or the use of open-ended entries while collecting data.

It is a less common practice to specify anomalous values than missing data in data dictionaries. Therefore, it becomes a vital task to detect and deal with anomalous data. The challenge then is mostly due to the difficulty of distinguishing normal and anomalous data. One may need to consider separate strategies to model anomalies for

different data types. For example, syntactic approaches can help to detect anomalies in date columns (e.g., detecting `Error` in a date column where the date values are represented using the ISO-8601 format), whereas numerical anomalies can be detected using statistical models rather than syntactic approaches.

2.2 Syntactic Type Inference

In syntactic type inference, the goal is to determine the type of a column based on the syntax of the data values in its entries. Numerous studies have attempted to tackle syntactic type inference. However, these methods provide a limited capability to detect missing and anomalous data, resulting in incorrect column type predictions as shown in Fig. 2.1 where data columns are labelled with the string type unless missing and anomalous data are detected.



Figure 2.1: **Normal**, **missing**, and **anomalous** values are denoted by green, yellow, and red, respectively.

We categorize syntactic type inference methods into two groups: (i) rule-based approaches and (ii) probabilistic approaches, which are discussed in detail below.

Rule-based Approaches

Table 2.1 presents an overview of the capabilities of existing rule-based syntactic type inference methods. As per the table, they all address the detection of basic types and

missing data, although only a limited number of missing data encodings is supported. However, they do not generally consider the detection of non-Boolean categorical variables, complex types and anomalous data.

Method	Basic Types	Non-Boolean Categorical	Complex Types	Missing Data	Anomalies
Bot	✓	✓		✓	
F#	✓		✓	✓	
hypoparsr	✓			✓	
messytables	✓			✓	
OpenML	✓	✓		✓	
readr	✓			✓	
TDDA	✓			✓	
Trifacta	✓		✓	✓	✓
Weka	✓	✓		✓	

Table 2.1: A summary of the rule-based type inference methods. Here, the presence of a ✓ sign indicates whether a method considers the corresponding issue.

Most existing rule-based methods, including F# (Petricek et al., 2016), hypoparsr (Döhmen et al., 2017), messytables (Lindenberg, 2017), PADS (Fisher and Gruber, 2005), readr (Wickham et al., 2017), Test-Driven Data Analysis (TDDA, Stochastic Solutions 2018) and Trifacta (Trifacta, 2018) combine regular expressions with certain rules for type inference. For example, Trifacta (2018) and its preceding versions (Raman and Hellerstein, 2001; Kandel et al., 2011; Guo et al., 2011) apply *validation functions* to a sample of data to infer types, e.g., assign the one validated for more than half of the non-missing entries as the column type, where a data value is validated using regular expressions (Kandel et al., 2011). When multiple types satisfy this criterion, the more specific one is chosen as the column type, e.g., an integer is assumed to be more specific than a float. Trifacta supports a comprehensive set of data types and provides an *automatic discrepancy detector* to detect errors in data (Raman and Hellerstein, 2001). However, in our experience, its performance on type inference can be limited on messy datasets (see Sec. 3.3.2 for our experimental results for type inference).

Fisher and Gruber (2005) and Fisher et al. (2008) have developed data description languages for processing ad hoc data sources (PADS) which enables generating a human-readable description of a dataset based on data types inferred using regular expressions. However, their focus is on learning regular expressions to describe a dataset, rather than classifying the data columns into known types.

Test-Driven Data Analytics (TDDA, Stochastic Solutions 2018) uses the Pandas CSV reader to read the data into a data frame. It then uses the Pandas dtypes attributes¹, to determine the data type of the columns. However, this leads to a poor type detection performance since the Pandas reader is not robust against missing data and anomalies, where only empty string, NaN, and NULL are treated as missing data unless the user specifies differently.

Petricek et al. (2016) propose another use of regular expressions with F#, where types, referred as *shapes*, are inferred based on a set of *preferred shape relations*. Such relations are used to resolve ambiguous cases where a data value fits multiple types. This procedure allows integrating inferred types into the process of coding, which can be useful to interact with data. However, it does not address the problems of missing and anomalous data comprehensively, where only three encodings of missing data, NA, #NA, and :, are supported. This may lead to poor performance on type inference when missing data is encoded differently, or there are anomalies such as error messages.

A number of software packages in R and Python can also infer data types. *messytuples* (Lindenberg, 2017) determines the most probable type for a column by weighting the number of successful conversions of its elements to each type. This can potentially help to cope with certain data errors; however, it might be difficult to find an effective configuration of the weights for a good performance. Moreover, it can not handle the *disguised missing data* values, e.g., -1 in a column of type integer, which can be misleading for the analysis. Döhmen et al. (2017) propose a CSV parser named *hypoparsr* that treats type inference as a parsing step. It takes into account a wide range of missing data encodings; however, it does not address anomalies, leading to poor performance on type inference (see Sec. 3.3.2 for our experimental results for type inference). *readr* (Wickham et al., 2017) is an R package to read tabular data, such as CSV and TSV files. However, in contrast to *hypoparsr*, a limited set of values are considered as missing data, unless the user specifies otherwise. Furthermore, it employs a heuristic search procedure using a set of matching criteria for type inference. The details regarding the criteria are given in Wickham and Grolemund (2016). The search continues until one criterion is satisfied, which can be applied successfully in certain scenarios. Whenever such conditions do not hold, the column type is assigned to string.

The methods discussed above support various data types including the Boolean type, which can be seen as a subtype of the categorical type; however, they do not consider **non-Boolean categorical variables** where there are more than two categor-

¹obtained with the function `pandas_tdda_type()`.

ical values. Consequently, they treat such columns as either integers or strings rather than categoricals. A limited number of works can identify the general categorical type (including the non-Boolean variables). Bot (Majoor and Vanschoren, 2018) reads the data by using the `Pandas.read_csv()` function and applies a set of heuristics to map the inferred data types to the types of categorical, date, float, integer and string. For example, if the `Pandas.read_csv()` function infers the type of a data column as integer and the data entries has two unique values, the column is labelled as categorical. Similarly, a data column initially labelled with the integer type is treated as a categorical variable when one of the following conditions is satisfied: (i) if the number of unique values is less than 11 and (ii) if the number of unique values is between 10 and a pre-defined value and the average of absolute distances between integers is lower than the average of integers. Bot uses similar heuristics to identify categorical variables where the possible values are encoded by strings. These heuristics would help to identify categorical variables in certain scenarios. However, we consider a more flexible solution based on a trained machine learning model in Chapter 4.

An alternative approach to Bot is to use the file conversion techniques that convert Comma-separated values (CSV) files to the Attribute Relation File Format (ARFF)². For example, `csv2arff`³ is used in OpenML to identify ARFF data types (date, nominal, numeric and string) and categorical values. Note that nominal is the ARFF term used to refer to categorical. It uses the Pandas library to parse a data file and employs a rule-based approach that is similar to Bot (e.g., a data column is labelled with the integer type if (i) all the data values are either missing data indicators or integers and (ii) the number of unique values except those detected as missing are higher than 10). Therefore, this method offers limited capability compared to a trained probabilistic model. Additionally, it does not distinguish between the float and integer types, as such columns are labelled with the ARFF numeric type. Weka⁴ provides another rule-based approach for CSV to ARFF conversion. For example, a data column is labelled as numeric if all the data values except those detected as missing can be converted to floating-point numbers. Note that it can only detect missing data encoded by the question mark symbol `?`, unless the user specifies differently. This conversion method suffers from similar issues as OpenML's converter.

²The details are available at https://waikato.github.io/weka-wiki/formats_and_processing/arff_stable/ [Accessed on 05/12/2020]

³The code is available at <https://github.com/openml/ARFF-tools/blob/master/csv-to-arff.py> [Accessed on 09/11/2020]

⁴The details are at https://waikato.github.io/weka-wiki/formats_and_processing/convert_csv_to_arff/ [Accessed on 05/12/2020].

Following the detection of categorical variables, CSV to ARFF conversion methods typically carry out identification of categorical values, where the goal is to identify the categorical values of a categorical variable. For example, OpenML's `csv2arff` first identifies missing data using the `isnan()` functions of the NumPy and math packages. Once this has been done, the remaining values are identified as the categorical values. Although these functions can help to handle standard missing data encodings, they do not handle anomalous data. The rule-based approach used in Weka suffers from similar issues as OpenML's converter.

Probabilistic Approaches

A few number of studies tackle type inference from a probabilistic perspective. Valera and Ghahramani (2017) and Vergari et al. (2019) propose probabilistic models for discovering statistical types including the categorical type. However, both models are tackling very different problems than the one that we address in Chapters 3 and 4. Valera and Ghahramani (2017) assume that the data contains only numerical values and that it does not have any missing data and anomalies. Given this they address the problem of making fine-grained distinctions between different types of continuous variables (real-valued data, positive real-valued data, and interval data) and discrete variables (categorical data, ordinal data, and count data). Similarly, Vergari et al. (2019) assume that each entry of a data column contains a numerical value or an explicit missing data indicator. The authors tackle *missing data imputation* rather than *missing data detection* and attempt to detect numerical outliers, but do not detect string anomalies in a column.

Eduardo et al. (2020) tackle detection and repair of cell-level outliers. The authors propose a deep generative model for tabular data consisting of numeric and categorical features. The idea is to extend Variational Autoencoder by introducing an additional component to the observation model per feature, which accounts for the corruptions in the data. Although this approach has been shown useful to detect and repair numerical outliers, it does not address how to handle string anomalies (e.g., error messages).

2.3 Semantic Type Inference

In semantic type inference, the goal is to go beyond the syntactic types and provide further insights into the data. A growing body of literature (Limaye et al., 2010; Zwickl-

bauer et al., 2013; Chen et al., 2019a) has investigated the use of knowledge graphs (KGs) for semantic type inference, where the task is to annotate each column of a dataset with an entity class in a KG. Limaye et al. (2010) propose a log-linear model based method to annotate a table in terms of the semantic column types, cell entities, and column relations, given an ontology of relevant information. For example, given a table that contains information about actors such as names and ages, the task is to find semantic column types of actor and age, the entities each row refers to, and the dependent relationship between two columns, i.e., one column denotes the age of an actor whose name is given in the other column. Zwicklbauer et al. (2013) focus only on semantic column type inference rather than solving the three tasks mentioned above collectively. The authors employ cell-to-entity matching and determine the column type through a majority voting on the annotated cell entities. A more sophisticated approach is proposed by Chen et al. (2019a) where a neural network is combined with KG lookups. The scope of the annotations of these methods are typically wider than the annotations considered by syntactic type inference methods. However, they are limited by the extent of the information held in the KG.

Chapter 3

Syntactic Type Inference

In the remainder of this chapter, we first motivate our work on syntactic type inference (Section 3.1). Then, we describe Probabilistic Finite-State Machines (PFSMs) and introduce our model (Section 3.2). Finally, we present the experiments and the results (Section 3.3), which is followed by a summary of our work and a discussion of the possible future research directions (Section 3.4).

3.1 Introduction

As we describe in Sec. 2.1, the goal in syntactic type inference is to infer the data type (e.g., date, float, integer and string) for each column in a table of data. Numerous studies have attempted to tackle the syntactic type inference task, including wrangling tools (Raman and Hellerstein, 2001; Kandel et al., 2011; Guo et al., 2011; Trifacta, 2018; Fisher and Gruber, 2005; Fisher et al., 2008), software packages (Petricek et al., 2016; Lindenberg, 2017; Stochastic Solutions, 2018; Döhmen et al., 2017; Wickham et al., 2017) and probabilistic approaches (Valera and Ghahramani, 2017; Vergari et al., 2019; Limaye et al., 2010) (see Sec. 2.2 for a detailed discussion of these methods). However, often they do not work very well in the presence of missing and anomalous data, which are commonly found in raw data sets due to the lack of a well-organized data collection procedure.

Consider the toy example given in Figure 2.1 (Sec. 2.2) which consists of three data columns with missing data encoded by `NA`, `-1` and `Null` and anomalous data valued `Error` and `&%&`. Such values make the type inference task challenging for the existing approaches, and need to be detected before further analysis. See Sec. 2.1 for an overview of missing and anomalous data.

Up to now, too little attention has been paid to the aforementioned problems by the data mining community. To this end, we introduce *ptype*, a probabilistic type inference method that can accurately infer the data type for a column of data even in the presence of missing and anomalous data. The proposed model is built upon PFSMs that are used to model known data types, missing and anomalous data. In contrast to the standard use of regular expressions, PFSMs have the advantage of generating weighted posterior predictions even when a column of data is consistent with more than one type model. Our method is shown to generally outperform existing type inference approaches for inferring data types, and also allows us to identify missing and anomalous data entries.

3.2 Methodology

This section consists of four parts. Sec. 3.2.1 gives background information on PFSMs used to model regular data types, missing data, and anomalies. Sec. 3.2.2 introduces our model that uses a mixture of PFSMs. Lastly, Sec. 3.2.3 and Sec. 3.2.4 describe respectively inference in and training of this model.

The data type, missing data, and anomalies can be defined in broad terms as follows: The data type is the common characteristic that is expected to be shared by entries in a column, such as integers, strings, IP addresses, dates, etc., while missing data denotes an absence of a data value which can be encoded in various ways, and anomalies refer to values whose types differ from the given column type or the missing type.

In order to model above types, we have developed PFSMs that can generate values from the corresponding domains. This, in turn, allows us to calculate the probability of a given data value being generated by a particular PFSM. We then combine these PFSMs in our model such that a data column \mathbf{x} can be annotated via probabilistic inference in the proposed model, i.e., given a column of data, we can infer column type, and rows with missing and anomalous values.

3.2.1 Probabilistic Finite-State Machines

Finite-State Machines (FSMs) are a class of mathematical models used to represent systems consisting of a finite number of internal states. The idea is to model a system by defining its states (including initial and final states), transitions between the states, and external inputs/outputs. FSMs have a long history going back at least to

Rabin and Scott (1959) and Gill (1962). A more recent overview of FSMs is given by Hopcroft et al. (2001).

In this study, we are interested in a special type of FSMs called Probabilistic Finite-State Machines (PFSMs) in which transitions between states occur with respect to probability distributions (Paz, 1971; Rabin, 1963). Vidal et al. (2005) discuss various PFSMs in detail. Following a similar notation, we define a PFSM as a tuple $A = (\theta, \Sigma, \delta, I, F, T)$, where θ is a finite set of states, Σ is a set of observed symbols (a given subset of characters in our case), $\delta \subseteq \theta \times \Sigma \times \theta$ is a set of transitions among states with respect to observed symbols, $I : \theta \rightarrow [0, 1]$ is the initial-state probabilities, $F : \theta \rightarrow [0, 1]$ is the final-state probabilities, and $T : \delta \rightarrow (0, 1]$ is the transition probabilities for the elements of δ . During each possible transition between states, a symbol is emitted. We denote such an event by a triple (q, α, q') , which corresponds to a transition from a state $q \in \theta$ to a state $q' \in \theta$ emitting a symbol $\alpha \in \Sigma$. Note that δ and T store respectively all the possible triples and their corresponding probabilities.

A PFSM has to satisfy certain conditions. First, the sum of the initial-state probabilities has to be equal to 1. Secondly, at each state q , it can either transition to another state $q' \in \theta$ and emit a symbol $\alpha \in \Sigma$, or stop at state q without emitting any symbol. This can be expressed mathematically as $F(q) + \sum_{\alpha \in \Sigma, q' \in \theta} T(q, \alpha, q') = 1$ for each state q , where $T(q, \alpha, q')$ represents the probability of a triple (q, α, q') , and $F(q)$ denotes the final-state probability of state q . Based on the definition given above, a PFSM can generate a set of strings, denoted by Σ^* .¹ For each string $s \in \Sigma^*$, we can calculate a probability that represents how likely it is for a given PFSM to generate the corresponding string.

Note that PFSMs resemble Hidden Markov Models (HMMs) except that we now have the final state probabilities. Recall that each state in an HMM has a probability distribution over the possible states that it can transition to. In PFSMs, each state also takes into account the probability of being a final state. Hence, the probability distribution is not only defined over the possible transitions to next states; it also includes the case of the current state being a final state. On the other hand, emissions are carried out similarly in PFSMs and HMMs: the observations are generated conditioned on the hidden states in HMMs; an observation is emitted through a transition in PFSMs since each transition is associated with a symbol. A detailed analysis of the link between PFSMs and HMMs can be found in Dupont et al. (2005).

One can develop PFSMs to represent types described previously and calculate the

¹ Σ^* denotes the set of strings a PFSM can generate by emitting multiple symbols.

probabilities for each observed data value. We now explain the corresponding PFSMs in detail.

Representing Types with PFSMs

Here, we show how a PFSM can be used to model a data type. We divide types into two groups: (i) primitive types consisting of integers, floats, Booleans and strings, and (ii) complex types such as IP addresses, email addresses, phone numbers, dates, postcodes, genders and URLs. The details regarding the implementation of the corresponding PFSMs can be found in Appendix A.1.

Consider integer numbers whose domain is $\{-\infty, \infty\}$. We can represent the corresponding PFSM as in the diagram given in Figure 3.1. The machine has two initial states, namely q_0 and q_1 , and one final state q_2 . Here, q_0 and q_1 respectively allow us to represent integer numbers with a sign (plus or minus), or without any sign.² The machine eventually transitions to the state q_2 , which stops with a stopping probability $F(q_2) = P_{stop}$. Otherwise, it transitions to itself by emitting a digit with an equal probability, $(1 - P_{stop})/10$. Similarly, we can develop a PFSM to represent each one of the other column types.

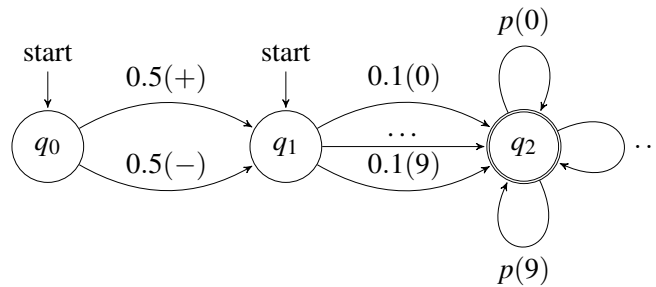


Figure 3.1: Graphical representation of a PFSM with states $\theta = \{q_0, q_1, q_2\}$ and alphabet $\Sigma = \{+, -, 0, 1, \dots, 9\}$ where p denotes $\frac{1 - P_{stop}}{10}$.

The PFSM for missing values can be developed by using a pre-defined set of codes such as $\{-1, -9, -99, \text{NA}, \text{NULL}, \text{N/A}, -, \dots\}$. It assigns non-zero probabilities to each element in this set. Note that the corresponding PFSM already supports a wide range of codes; however, it can be easily extended by the user to incorporate other known missing data encodings, leading to semi-automated operation.

In order to model anomalies, we adapt the idea of *X-factor* proposed by Quinn et al. (2009). We define a machine with the widest domain among all the PFSMs that

²The transitions from state q_1 allow the emission of a zero, which means that numbers like -007 can be emitted. If this is not desired, one can adjust the PFSM in Figure 3.1 to not emit a leading 0.

supports all possible characters. This choice of PFSM lets the probabilistic model become more robust against anomalies since it assigns probabilities to data values that do not belong to any of the PFSMs representing the data types. Note that as this PFSM covers a wider domain, it will assign lower probabilities to known data types than the specific models.

Constructing PFSMs for complex types might require more human engineering than the other types. We reduce this need by building such PFSMs automatically from corresponding regular expressions. We first convert regular expressions to FSMs by using the greenery library³ (see the function named `to_fsm()`). We then build their probabilistic variants, where the parameters are assigned equal probabilities. Note that these parameters can be updated with the training.

3.2.2 The Proposed Model

We propose a new probabilistic mixture model with a noisy observation model, allowing us to detect missing and anomalous data entries. Our model first generates a column type from a set of possible regular data types. This is followed by a “deficiency” process that can potentially change the data type of each row. Consequently, each row might have a different type rather than the generated column type. The observation model then generates a data value for each entry according to its type. We now introduce our notation to represent this process.

We assume that a column of data $\mathbf{x} = \{x_i\}_{i=1}^N$ has been read in, where each x_i denotes the characters in the i^{th} row. We propose a generative model with a set of latent variables $t \in \{1, 2, \dots, K\}$ and $\mathbf{z} = \{z_i\}_{i=1}^N$, where t and z_i respectively denote the data type of a column and its i^{th} row. Here, N is the number of rows in a data column, and K is the number of possible data types for a column. We also use the additional missing and anomaly types, denoted by m and a respectively, and described above. Note that z_i can be of type m or a alongside a regular data type, i.e., $z_i \in \{1, 2, \dots, K, m, a\}$. This noisy observation model allows a type inference procedure robustified for missing and anomalous data values.

³The library is available at <https://github.com/qntm/greenery> [Accessed on 09/02/2021].

Hence the model has the following generative process:

$$\begin{aligned} \text{column type } t &\sim \mathcal{U}(1, K), \\ \text{row type } z_i &= \begin{cases} t & \text{with probability } \pi_t^t, \\ m & \text{with probability } \pi_t^m, \\ a & \text{with probability } \pi_t^a, \end{cases} \\ \text{row value } x_i &\sim p(x_i|z_i), \end{aligned}$$

where Π and $p(x_i|z_i)$ are respectively the model parameter and the observation model. \mathcal{U} denotes a discrete Uniform distribution. Here $\pi_t^t + \pi_t^m + \pi_t^a = 1$ for each column type t . Since entries are often expected to be of a regular data type rather than missing or anomaly types, we favour regular types during inference by using lower coefficients for missing and anomaly types, i.e., $\pi_t^m < \pi_t^t$ and $\pi_t^a < \pi_t^t$. These weight parameters Π are assumed to be fixed and known. Even though one can also learn the weights, which can help us adapt the model to how noisy data is, this may not be vital as long as the coefficients of the regular types are larger than the others. Note that the model might treat clean data entries as missing or anomalous when the corresponding coefficients are larger than the coefficients of the regular type, as this would encourage the probabilities assigned by the missing or anomaly type (e.g., `apple` might be treated as an anomaly rather than a string).

We use a uniform distribution for column types as in most cases we do not have any prior information regarding the type of a column that would allow us to favour a particular one. To represent the conditional distribution $p(x_i|z_i)$, we have developed PFSMs as described above.

3.2.3 Inference

Given a data column \mathbf{x} , our initial goal is to infer the column type t , which is cast to the problem of calculating the posterior distribution of t given \mathbf{x} , denoted by $p(t|\mathbf{x})$. Then, we assume that each row can be of three types: (i) the same as the column type, (ii) the missing type, and (iii) the anomaly type. In order to identify missing or anomalous data entries, we calculate the posterior probabilities of each row type, namely $p(z_i|t, \mathbf{x})$. In this section, we now briefly discuss the corresponding calculations.

Assuming that entries of a data column are conditionally independent given t , we

can obtain the posterior distribution of column type t as follows:

$$p(t = k|\mathbf{x}) \propto p(t = k) \prod_{i=1}^N \left(\pi_k^k p(x_i|z_i = k) + \pi_k^m p(x_i|z_i = m) + \pi_k^a p(x_i|z_i = a) \right), \quad (3.1)$$

which can be used to estimate the column type t , since the one with maximum posterior probability is the most likely data type corresponding to the column \mathbf{x} .

As per equation 3.1, the model estimates the column type by considering all the data rows, i.e., having missing data or anomalies does not confuse the type inference. Note that such entries would have similar likelihoods for each column type, which allows the model to choose the dominant data type for regular entries.

Following the inference of column type, we can also identify entries of \mathbf{x} which are more likely to be missing or anomalies rather than the inferred type. For this, we compare the posterior probabilities of each row type z_i given $t = k$ and x_i , namely $p(z_i = j|t = k, x_i)$, which can be written as:

$$p(z_i = j|t = k, x_i) = \frac{\pi_k^j p(x_i|z_i = j)}{\sum_{\ell \in \{k, m, a\}} \pi_k^\ell p(x_i|z_i = \ell)}. \quad (3.2)$$

Complexity Analysis

The computational bottleneck in the inference is the calculation of $p(\mathbf{x}|t = k)$ for each type k , which is the calculation of the probability assigned for a data column \mathbf{x} by the k^{th} PFSM. Note that this can be carried out by taking into account the counts of the unique data entries, for efficiency. Denoting the u^{th} unique data value by x_u , we need to consider the complexity of calculating $p(x_u|t = k)$ which can be done via the PFSM Forward algorithm. Each iteration of the algorithm has the complexity of $O(H_k^2)$ where H_k is the number of hidden states in the k^{th} PFSM. As the number of iterations equals to the length of x_u denoted by L_u , the overall complexity of the inference becomes $O(UKH^2L)$, where U is the number of unique data entries, K is the number of types, L is the maximum length of data values, and H is the maximum number of hidden states in the PFSMs.

3.2.4 Training of the Model

The aim of the training is to tune the probabilities assigned by the PFSMs so that column types are inferred accurately. Given a set of columns and their annotated column

types, the task is to find the parameters of the PFSMs (i.e., the initial-state, the transition, and the final-state probabilities) that allow the “correct” machine to give higher probabilities to the observed entries. This is crucial as multiple PFSMs can assign non-zero probabilities for certain strings, e.g., 1, True, etc.

We employ a discriminative training procedure on our generative model, as done in discriminative training of HMMs (Bahl et al., 1986; Jiang, 2010; Brown, 1987; Nádas et al., 1988; Williams and Hinton, 1991). This is shown to be generally superior to maximum likelihood estimations (Jiang, 2010) since a discriminative criterion is more consistent with the task being optimized. Moreover, it allows us to update not only the parameters of the “correct” PFSM but also the parameters of the other PFSMs given a column of data and its type, which in turn helps the correct one to generate the highest probability.

We choose $\sum_{j=1}^M \log p(t^j | \mathbf{x}^j)$ as the objective function to maximize, where \mathbf{x}^j and t^j respectively denote the j^{th} column of a given data matrix X and its type, and M is the number of columns. We then apply Conjugate Gradient algorithm to find the parameters that maximize this objective function (please see Appendix A.3 for detailed derivations of the gradients).

We study different parameter settings for our model. We first explore tuning the parameters by hand to incorporate certain preferences over the types, e.g., Boolean over integer for 1. Then, we learn the parameters via the discriminative training described above where the parameters are initialized at the hand-crafted values. Note that due to the absence of explicit labels for the missing and anomaly types, these are not updated from the hand-crafted parameters. We have also employed the training by initializing the parameters uniformly. However, we do not report these results as they are not competitive with the others.

As the PFSMs are generative models, it would be possible to train them *unsupervised*, to maximize $\sum_{i,j} \log p(x_i^j)$, where $p(x_i^j)$ is defined as a mixture model over all types (including missing and anomaly) for the i th row and j th column. The component PFSMs could then be updated using the Expectation-Maximization (EM) algorithm. However, such training would be unlikely to give as good classification performance as supervised training.

3.3 Experiments

In this section, we first in Sec. 3.3.1 describe the datasets and evaluation metrics used, and then in Sec. 3.3.2 compare *ptype* with competitor methods on two tasks: (i) column type inference, and (ii) type/non-type inference. Note that type/non-type inference refers to the task of identifying missing data and anomalies which are collectively labeled as non-type. Therefore, the task is to correctly classify each entry as either type or non-type. Lastly, we present a qualitative evaluation of our method on challenging cases in Sec. 3.3.3. The goal of our experiments is to evaluate (i) the robustness of the methods against missing data and anomalies for column type inference and (ii) the effectiveness of type/non-type inference. These are evaluated both quantitatively (Sec. 3.3.2) and qualitatively (Sec. 3.3.3). We release our implementation of the proposed method at <https://github.com/alan-turing-institute/ptype>.

3.3.1 Experimental Setup

We have trained *ptype* on 25, and tested on 43 data sets obtained from various sources including UCI ML⁴, data.gov.uk, ukdataservice.ac.uk, and data.gov. The data types were annotated by hand for these sets. We also annotated each dataset in terms of missing data and anomalies, by using the available meta-data, and checking the unique values.

On column type inference, we compare our method with F# (Petricek et al., 2016), *hypoparsr* (Döhmen et al., 2017), *messytables* (Lindenberg, 2017), *readr* (Wickham et al., 2017), TDDA (Stochastic Solutions, 2018) and *Trifacta* (2018). Note that some of the related works are not directly applicable to this task, and these are not included in these experiments. For example, we are not able to use Raman and Hellerstein (2001), Kandel et al. (2011) and Guo et al. (2011) as they are not available anymore. However, we use their latest version *Trifacta* in our experiments. We also exclude the PADS library (Fisher and Gruber, 2005; Fisher et al., 2008), since it does not necessarily produce columns and their types (see Appendix A.4 for an example). The methods proposed by Valera and Ghahramani (2017) and Vergari et al. (2019) are also not applicable to this task. First, they do not consider data types of Boolean, string, date. Secondly, they only address integer and float columns that do not contain any non-numerical missing data or anomalies, which are commonly found in real-world datasets. Note that Vergari et al. (2019) do not address missing data detection but

⁴<https://archive.ics.uci.edu/ml/datasets.html>

missing data imputation, and can only handle numerical outliers but not non-numerical outliers, whereas Valera and Ghahramani (2017) do not address these questions at all. Lastly, we exclude the method presented by Limaye et al. (2010) as their goal is to infer semantic entity types rather than syntactic data types.

On type/non-type inference, we compare our method with Trifacta only, as it is the leading competitor method on column type inference, and the others do not address this task comprehensively.

Data Sets

We have conducted experiments on the data sets chosen according to two criteria: (i) coverage of the data types, and (ii) data quality. We consider five common column types in our experiments: Boolean, date, float, integer, and string. Table 3.1 presents the distribution of the column types found in our data sets. Any other columns not conforming to the supported data types are omitted from the evaluations. Secondly, we have selected messy data sets in order to evaluate the robustness against missing data and anomalies. As per Fig. 3.2, the fraction of the non-type entries in the test datasets can be as large as 0.56, while the average fraction is 0.11. Note that available data sets, their descriptions and the corresponding annotations can be accessed via <https://goo.gl/v298ER>.

	Column Type					Total
	Boolean	Date	Float	Integer	String	
Training	75	49	99	257	309	789
Test	43	40	53	240	234	610

Table 3.1: Histogram of the column types observed in the training and the test data sets.

Evaluation Metrics

For column type inference, we first evaluate the overall accuracy of the methods on type inference by using the accuracy. However, this may not be informative enough due to the imbalanced data sets. Note that the task of type inference can be seen as a multi-class classification problem, where each column is classified into one of the possible column types. In order to measure the performance separately for each type, we follow

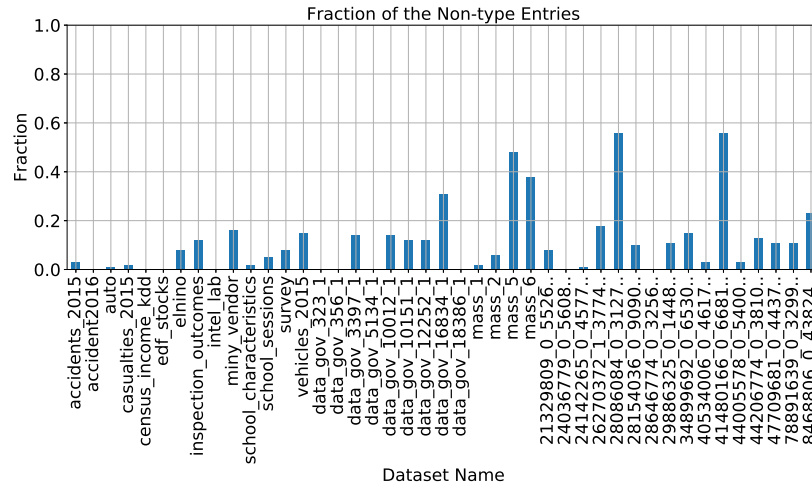


Figure 3.2: Fraction of the non-type entries in a dataset, calculated by aggregating over its columns. Note that ‘overall’ denotes the fraction after aggregating over the datasets.

a *one-vs-rest* approach. In such cases a common choice of metric is the Jaccard index J (see e.g., Hand et al. (2001, sec 2.3)) defined as $TP/(TP + FP + FN)$, where TP, FP, and FN respectively denote the number of True Positives, False Positives and False Negatives. (Note that *one-vs-rest* is an asymmetric labelling, so True Negatives are not meaningful in this case.)

To measure the performance on type/non-type inference, we report Area Under Curve (AUC) of Receiver Operating Characteristic (ROC) curves, as well as the percentages of TPs, FPs, and FNs. Note that here we denote non-type and type entries as Positive and Negative respectively.

3.3.2 Quantitative Results

We present quantitative results on two tasks: (i) column type inference, and (ii) type/non-type detection. In column type inference, we evaluate the performance of the methods on detecting data types of columns and investigate their scalability. Then, in type/non-type detection we evaluate their capability of detecting missing data and anomalies.

Column Type Inference

We present the performance of the methods in Table 3.2, which indicates that our method performs better than the others for all types, except for the date type where it is slightly worse than Trifacta. These improvements are generally due to the robustness of our method against missing data and anomalies. In the table ptype denotes the

discriminatively trained model, and `pctype-hc` the version with hand-crafted parameters.

Notice that the discriminative training improves the performance, specifically for Boolean and integer types. This shows that the corresponding confusions can be reduced by finding more optimal parameter values, which can be difficult otherwise. Note that the training has a slightly negative effect on float and string types, but it still performs better than the other methods. These changes can be explained by the fact that the cost function aims at decreasing the overall error over columns rather than considering individual performances, as we were concerned with the overall performance. However, one could modify the cost function to treat individual performances separately. A natural approach then would be to use different weights per type in the cost function, e.g., assigning weights based on the class distribution as in the *cost-efficient learning*.

	Method						
	F#	messytables	readr	TDDA	Trifacta	pctype-hc	pctype
Overall Accuracy	0.73	0.72	0.69	0.61	0.90	0.92	0.93
Boolean	0.55	0.56	0.00	0.00	0.49	0.75	0.83
Date	0.35	0.17	0.10	0.00	0.68	0.67	0.67
Float	0.60	0.57	0.59	0.42	0.87	0.93	0.91
Integer	0.55	0.55	0.57	0.46	0.88	0.85	0.88
String	0.61	0.61	0.58	0.51	0.83	0.90	0.89

Table 3.2: Performance of the methods using the Jaccard index and overall accuracy, for the types Boolean, Date, Float, Integer and String.

Figure 3.3 shows normalized confusion matrices for the methods, normalized so that a column sums to 1. This shows that all the methods tend to infer other column types as string even if they are not. However, `pctype` has few of such confusions, especially when the true type is Boolean or float.

It is noticeable from Table 3.2 that dates are difficult to infer accurately. Detailed inspection shows that this is due to non-standard formats used to denote dates. We could improve the PFSM for dates in `pctype` to include such formats, but we have not done so, to avoid optimizing on test datasets. The performance of Trifacta on dates can be explained by the engineering power behind the tool and indicates its capability to represent non-standard formats using validation functions.

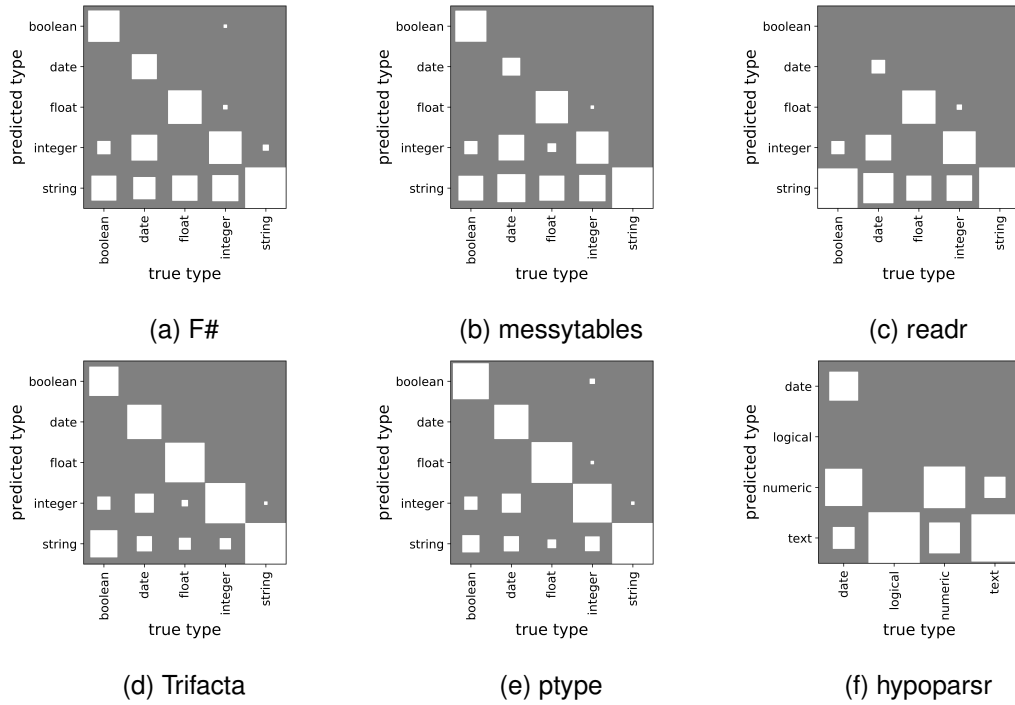


Figure 3.3: Normalized confusion matrices for (a) F#, (b) messytables, (c) ptype, (d) reader, (e) Trifacta, and (f) hypoparsr plotted as Hinton diagrams, where the area of a square is proportional to the magnitude of the entry.

To determine whether the column type predictions of ptype and Trifacta are significantly different, we apply the McNemar’s test (see e.g., Dietterich 1998), which assumes that the two methods should have the same error rate under the null hypothesis. We compute the test statistic $(|n_{01} - n_{10}| - 1)^2 / (n_{01} + n_{10})$, where n_{01} and n_{10} denote the number of test columns misclassified by only Trifacta, and by only ptype respectively. In our case, n_{01} and n_{10} are respectively equal to 19 and 6, which results in a statistic of 5.76. If the null hypothesis is correct, then the probability that this statistic is greater than 3.84 is less than 0.05 (Dietterich, 1998). Thus this result provides evidence to reject the null hypothesis, and confirms that the methods are statistically significantly different from each other.

The large performance gap for Booleans suggests that our method handles confusions with integers and strings better. Analysis shows that such confusions occur respectively in the presence of $\{0, 1\}$, and $\{\text{Yes}, \text{No}\}$.⁵ We further note that F#, messytables, and readr perform similarly, especially on floats, integers, and strings; which is

⁵We assume that a data column where the entries are valued as Yes or No is more likely to be a Boolean column than a string. We have also confirmed these cases with the corresponding metadata whenever available, and have carefully annotated our datasets in terms of data types.

most likely explained by the fact that they employ similar heuristics.

Since hypoparsr names column types differently, except for the date type, we need to rename the annotations and re-evaluate the methods in order to compare them with hypoparsr. It refers to Boolean and string respectively as logical and text. Moreover, integer and float are grouped into a single type called numeric. The resulting evaluations, which are reported in Table 3.3, shows a similar trend to as before in that our method performs better. However, we see some variations, which result from the fact that we now evaluate on a smaller number of data sets since hypoparsr, which is said to be designed for small sized files, was able to parse only 33 out of the 43 test data sets.⁶ This left us 358 columns including 29 date, 21 logical, 159 numeric, and 149 text. Lastly, we observe that hypoparsr results in many confusions by inferring the type as integer whereas the true type is text. Such cases mostly occur when the data values consist of a combination of numeric and text, e.g., ID and address columns.

	Method							
	F#	hypoparsr	messytables	readr	TDDA	Trifecta	ptype-hc	ptype
Overall Accuracy	0.65	0.66	0.65	0.63	0.60	0.88	0.96	0.95
Date	0.31	0.31	0.17	0.07	0.00	0.62	0.66	0.66
Logical	0.27	0.00	0.29	0.00	0.00	0.14	0.88	0.88
Numeric	0.43	0.52	0.43	0.45	0.39	0.88	0.94	0.93
Text	0.56	0.54	0.57	0.55	0.52	0.80	0.94	0.93

Table 3.3: Performance of the methods using the Jaccard index and overall accuracy, for the types Date, Logical, Numeric and Text.

Next, we discuss the scalability of our method.

Scalability

We describe the complexity of the inference in our model in Section 3.2.3. Here, we demonstrate its scaling by measuring the time it takes to infer the column types for each test dataset.

Recall that we do not take into account pre-processing steps to calculate the complexity of the inference as we assume that the data has already been read in, and the

⁶This was using the default settings, but Till Döhmen (pers. comm.) advised against changing the limit.

unique values have already been detected. Therefore, the complexity is $O(UKH^2L)$, where U is the number of unique data entries, K is the number of data types, H is the maximum number of hidden states in the PFSMs, and L is the maximum length of data values. Notice that the complexity depends on data through U and L , and does not necessarily increase with the number of rows. In fact, it grows linearly with the number of unique values assuming L is constant. As shown in Fig. 3.4, the runtime for `pctype` is upper bounded by a line $c_0 + c_1U$, where c_0 is a small constant. The runtime thus scales linearly with the number of unique data entries U , handling around 10K unique values per second. The variations can be explained by changes in L .

We also report the size of the datasets and the times the methods take in Appendix A.5. We have observed similar performance with `messytables`, whereas `readr` and `TDDA` seem much faster even though they do not only predict the data types but also parse a given dataset (averaging the times reported in Table A.3 per dataset, we obtain 0.46 seconds for `pctype`, 0.37 seconds for `messytables`, 0.02 seconds for `readr` and 0.0002 seconds for `TDDA`). On the other hand, `hypoparsr` takes 18.23 seconds on average, which is much longer compared to the others. Lastly, we measure the processing times for `Trifacta` via command line. We have observed that `Trifacta` takes 0.99 seconds which indicates that it is faster than `hypoparsr`, but slower than the other methods (on average).

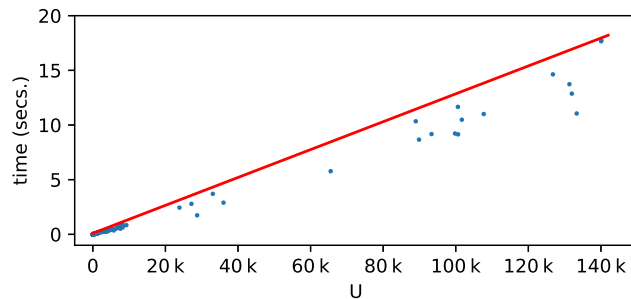


Figure 3.4: The time in seconds taken to infer a column type with `pctype`, as a function of the number of unique data entries U in the column. Also shown is the line $c_0 + c_1U$, where c_0 is a small constant.

Type/Non-type Inference

`Trifacta` labels each entry in a data table either as type or non-type, whereas our model presents a probability distribution over the two labels. One could apply a threshold on these probabilities in order to assign a label to each entry. Here, we demonstrate

how the methods behave under different thresholds. We aggregate the entries of each dataset over its columns, and compute the ROC curve for each method.

Figure 3.5 presents the difference $AUC(\text{ptype}) - AUC(\text{Trifacta})$ per dataset. Note that we exclude five datasets as the corresponding AUCs are undefined due to the definition of True Positive Rate ($\frac{TP}{TP+FN}$). This becomes undefined since the denominator becomes zero when both TP and FN are equal to zero, which occurs naturally when a dataset does not contain any missing data and anomalies. The average of AUCs of the remaining datasets are respectively 0.77 and 0.93 for Trifacta and ptype. To compare these two sets of AUCs, we apply a paired t-test, which results in the t-statistic of 4.59 and p-value of 0.00005. These results reject the null hypothesis that the means are equal, and confirm that they are significantly different.

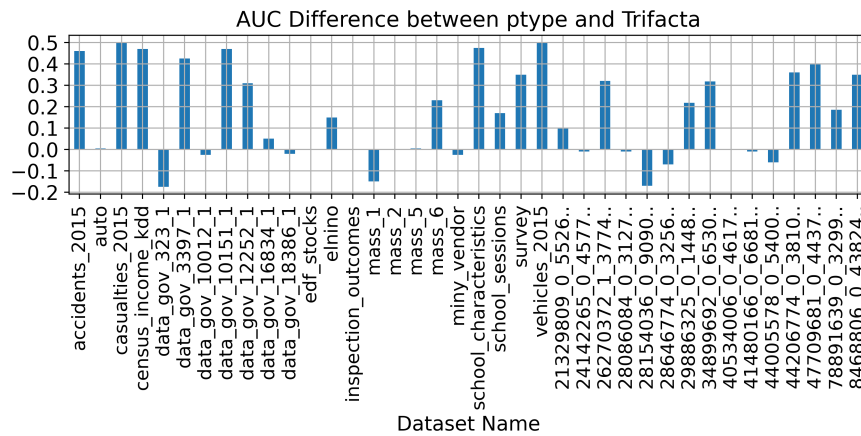


Figure 3.5: $AUC(\text{ptype}) - AUC(\text{Trifacta})$ plotted for each test dataset.

Figure 3.6 denotes the ROC curves plotted for some of the test datasets. Notice that Trifacta produces only one point for each dataset regardless of the threshold used, whereas ptype provides a more flexible solution. For example, Trifacta leads to a TPR of 0.57 and a FPR of 0.05 on the mass_6 dataset. However, with ptype we can obtain a much higher True Positive Rate in exchange for a slightly reduced False Positive Rate.

Lastly, we compare Trifacta and ptype in terms of percentages of TPs, FPs, and FNs which are presented in Table 3.4, where the labels for ptype are generated by applying a threshold of 0.5 on the posterior distributions. Note that here we aggregate the predictions over the datasets. As per the table, ptype results in a higher number of FPs than Trifacta, but Trifacta produces a higher number of FNs and a lower number of TPs than ptype. Note that here we denote non-type and type entries as Positive and Negative respectively. This indicates that ptype is more likely to identify non-types (TPs) than Trifacta, but it can also label type entries as non-types (FPs) more often.

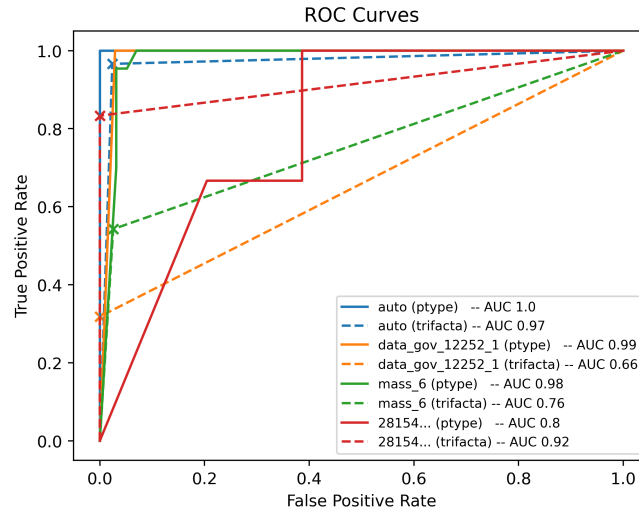


Figure 3.6: ROC curves plotted for some of the test datasets.

However, the overall performance of ptype is better than Trifacta, as we have also observed in the AUCs.

Method	FPS	FNs	TPs
Trifacta	0.67	3.96	1.57
ptype	1.13	0.20	5.34

Table 3.4: The percentages of FPS, FNs, and TPs for Trifacta and ptype on type/non-type detection.

3.3.3 Qualitative Results

We now give some examples of predicting missing and anomalous data.

Missing Data: We support an extensive range of values that are used to denote missing data. Note that multiple such encodings can be detected at the same time. Consider a T2Dv2 dataset⁷ where missing entries are denoted by encodings such as NULL and n/a. Our method can successfully annotate such entries as shown in Figure 3.7.

Next, we show how our model approaches unseen missing data encodings which are not explicitly considered as missing data in our model, but can be handled with the anomaly type. For example, “Time of Incident” column of Reported Taser 2015 data

⁷34899692_0_6530393048033763438.csv

country	buying rate	selling rate
 	NULL	NULL
austria	0.08546	refer to euro
bahamas	n/a	1.0254
trinidad & tobago	0.1416	0.1732

Figure 3.7: Annotating a subset of a T2D data set using *ptype* as **normal**, **missing**, and **anomalous** entries, denoted by green, yellow, and red, respectively.

set is expected to contain date values. However, some entries have the value `Unknown`. Thanks to the PFSM for anomaly type, our model detects such encodings as anomalies as long as the “correct” column type is not string, resulting in a better type inference performance.

Another advantage of *ptype* is that it can detect missing data even if their types fit the column type. Consider an integer-typed column `Stake` in the `Rodents` data set where `-99` is used to denote missing entries. Our method flags those entries as missing instead of treating them as regular integers since the missing type accepts `-99`. Similarly, we are able to detect string-valued missing data in string-typed columns. When the column type, however, is not string, our model may result in a number of false alarms by flagging normal entries as missing data. For example, integer values of `-1`, `-99`, etc. can also represent normal data instances. We could investigate this using methods as in Qahtan et al. (2018), and develop missing data models specific to each data type, in order to improve this issue.

Lastly, we compare *ptype* and `Trifacta`, the leading competitor method, to give insight into Table 3.4 in terms of how they handle missing data. Consider the “`Substantial_growth_of_knowledge_skills`” column of the `mass_6` dataset which consists of floating-point numbers. However, most of the 3148 entries are non-types, i.e., empty entries, `-`, `N/A`, and `NA` which occur 780, 470, 1063, and 424 times respectively. Such non-type entries are labeled correctly by *ptype*, whereas `Trifacta` can only classify the empty entries correctly as non-type. The remainder of the non-type entries are considered to be valid type entries, since they conform with the column type which is inferred as string by `Trifacta`. Note that this confusion is high likely due to the low number of floats in the column. Consequently, `Trifacta` results in a higher percentage of FNs and a lower percentage of TPs than *ptype*, as shown in Table 3.4.

Anomalies: As mentioned earlier, we can also use *ptype* to detect anomalies. Our model flags such entries automatically since the anomaly model covers a wide range of values including those that are not supported by the column type.

Figure 3.7 shows the capability of detecting anomalies when the column type is string. As we do not have the character & in the alphabet of the PFSM for strings, the anomaly machine allows us to detect the anomalies in the “country” column. Similarly, the characters “refer to euro” are not supported by the PFSM for integers, letting us detect the corresponding entry as anomalous. Moreover, we can separately detect the anomalous and missing data as in the “selling rate” column.

Interestingly, we notice that the question mark character ? is used to express the doubt about the collected data in the HES data set, where a data entry contains the value of 6?. We can also see that missing data encodings not incorporated to our missing data model such as NOT AVAILABLE, ?? (double question marks), and -, are detected as anomalies. Note that the user can easily let our model treat such values as missing data rather than anomalies by including them in the known missing data encoding list.

Next, we illustrate the extent of cases in which ptype results in FPs and compare it with Trifacta. For example, the “CAUSE_NAME” column in the data_gov_323_1 dataset consists of data values such as Cancer, Stroke, and Suicide etc. Here, ptype and Trifacta infer the column type as string, and label such entries correctly as type entries. However, Alzheimer’s disease and Parkinson’s disease are misclassified as non-types by ptype (1,860 FPs out of 13,260 entries) as our string model does not support the apostrophe. To handle this, we could include ’ in the corresponding alphabet, but we also find it helpful to detect “true” non-type entries having that character. We believe that such cases should be left to users with domain knowledge as they can easily extend the alphabet of the string type.

We now discuss some other aspects of our model, such as ambiguities that can occur, and how they can be handled; and failure cases which can be difficult to avoid.

The Limitations of Our Work

Ambiguous Cases: In certain cases, the posterior probability distribution over types is not heavily weighted on a particular type. For example, consider a column of data that contains values of NULL and 1. This could fit multiple PFSMs as 1 can either be an integer, a float, a string, or a Boolean value. However, we have assumed that 0 and 1 are more likely to be indicators of Booleans and have thus tuned the parameters of the corresponding machine such that the probabilities associated with 0 and 1 are slightly higher than the ones assigned by the other machines. This leads to a posterior probability distribution with values of 0.29, 0.26, and 0.44 respectively for integers,

floats, and Booleans. One can also exploit the probabilistic nature of our model to treat such ambiguous cases differently. Instead of directly taking the type with the highest posterior probability as the column type, one can detect such ambiguities, and then exploit user feedback to improve the decision.

A uniformly distributed posterior distribution over types is observed when all of the entries of a column are assigned zero probabilities by the PFSMs of the regular data types. This is not surprising as we only support a limited set of characters in the regular machines, i.e., the widest alphabet among the regular data types, which is of the string type, consists of the letters, digits, and a set of punctuations. For example, “per capitagdp (us\$)[51]” column of a T2D data set⁸ has values such as \$2949.57. Similarly, the 23rd column of the fuel data set contains values such as “(3.0L) ”. Note that the anomaly type still assigns positive probabilities to these entries, as its alphabet includes all the possible characters. However, when its weight π_t^a is the same regardless of the type, the corresponding posterior probabilities become equal. We leave such cases with high uncertainty for the user to handle as they can often be resolved by extending the alphabet of the string type.

Failure Cases: We now present two cases for which ptype-hc fails to infer the column types correctly. For example, consider a column (“BsmtHalfBath” of Housing Price data set) which denotes the number of half bathrooms in the basement of houses, consisting of the values in $\{0, 1, 2\}$. In this case, ptype-hc puts higher posterior probability on the Boolean type whereas the actual type is integer. This may not be surprising, considering the fact that 0 and 1 have higher probabilities of being a Boolean, and 2, occurring only twice out of 1460 entries, is treated as an anomaly. However, ptype is able to correct this failure thanks to the discriminative training. Note that the competitor methods fail in this case.

After the evaluations, we have discovered a set of cases we have not considered in the beginning. For example, several Boolean columns of the Census Income KDD data set have leading whitespace as in “ No”, “ Yes”. Our model infers the types of these Boolean columns as string since such whitespace is not considered in the Boolean type. In order to avoid optimizing the model on the test sets, we have not addressed such cases. However, they can easily be handled by updating the corresponding PFSMs to include the whitespace. Note that the other methods also detect the column types as string in these cases.

There are cases of non-standard dates we do not currently handle. For example,

⁸24036779_0_5608105867560183058.csv

dates are sometimes divided into multiple columns as day, month, and year. Our model detects day and month columns separately as integers. One could develop a model that checks for this pattern, making use of constraints on valid day, month and year values.

User Interactions for ptype

Above we discuss the limitations of our model and how they can be handled through user feedback. Our implementation of ptype supports a number of user interactions. These are summarized below (see the demonstrations at

<https://github.com/alan-turing-institute/ptype/blob/develop/notebooks>).

First, we address the issue of obtaining incorrect column type predictions. In such cases, our implementation allows the user to select an alternative type for a column. Note that selecting a new column type may also change the row type predictions, as row type inference is carried out conditioned on the column type.

Secondly, we consider the problems of having incorrect missing data predictions, such as treating normal data values as missing and treating missing data as valid. These issues occur due to a mismatch between the list of known missing data encodings supported by the PFSM for the missing type and the list of values used to encode missing data in a dataset. To handle such issues, we let the user modify the missing data encoding list by adding a new encoding or removing an existing one. Note that the list can either be modified for a specific column or all the columns at the same time. A similar mechanism is also provided for handling incorrect anomalous data predictions.

Thirdly we provide a capability to extend the alphabet of the PFSM for the string type. The user may need this capability when the default alphabet does not include a specific character, e.g., the character of \$ in the “per capitagdp (us\$)[51]” column of a T2D dataset containing values such as \$2949.57 (see the discussion in the limitations of our work above).

It is possible to extend the user interactions discussed above. For example, we can determine the columns of a dataset that ptype is unsure about based on the posterior probability distributions and suggest the user to check the assigned predictions only for those columns. Similarly, one can rank data columns based on uncertainties and let the user correct predictions in an order, which can assist the user in reviewing the predictions especially when the number of columns is high.

3.4 Summary

We have presented `pctype`, a probabilistic model for column type inference that can robustly detect the type of each column in a given data table, and label non-type entries in each column. The proposed model is built on PFSMs to represent regular data types (e.g., integers, strings, dates, Booleans, etc.), missing and anomaly types. An advantage of PFSMs over regular expressions is their ability to generate weighted posterior predictions even when a column of data is consistent with more than one type model. We have also presented a discriminative training procedure which helps to improve column type inference. Our experiments have demonstrated that we generally achieve better results than competitor methods on messy data sets.

Possible directions for future work include extending the supported data types, such as non-Boolean categorical data which would be treated either as integer or string by `pctype` (addressed in Chapter 4), etc.; developing subtypes, e.g., for Booleans expecting either `True` and `False`, or `yes` and `no`; and improving anomaly detection for string-typed data by addressing semantic and syntactic errors.

Chapter 4

Inferring the Type and Values of Categorical Variables

In this chapter, we first describe our motivation in this work (Sec. 4.1) and the proposed methodology (Sec. 4.2). Then, we present the related experiments (Sec. 4.3), which is followed by a discussion of our work and its potential extensions (Sec. 4.4).

4.1 Introduction

In Chapter 3, we consider five main data types, namely Boolean, date, float, integer and string, and propose a probabilistic model that can be used to annotate a data column with one of these types based on the syntax of its data entries. This approach provides limited support for the categorical type as non-Boolean categorical variables are treated either as integers or strings rather than categoricals. For example, the “Class Name” and “Rating” columns¹ in Fig. 4.1 are two examples of non-Boolean categorical variable and would be respectively annotated with the string and integer types by syntactic type inference methods including *ptype*. Therefore, the user needs to transform these data columns into categorical manually, which can be tedious and time-consuming. In this work, we present a probabilistic method called *ptype-cat* that can detect the general categorical type (including non-Boolean variables). In addition, we identify the values associated with the corresponding categorical variable by adapting *ptype*. By combining these two methods, we propose an alternative approach that can eliminate the need for manual work.

¹The data is subsampled from the Women’s Clothing E-Commerce dataset, which is publicly available at <https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews>.

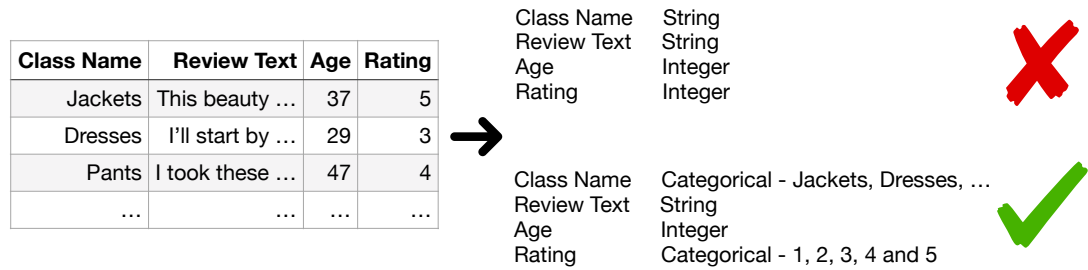



Figure 4.1: Overview figure.

A problem for the identification of the categorical type from the data in a column is that the values may be *encoded* in different ways. For the “Class Name” column these are encoded as *strings*, but the values might also be encoded as *integers* as in the “Rating” column, e.g., 1, 2, 3, 4 and 5. However, in both cases, the limited number of possible values taken on by a categorical variable is the key to its identification. Note that the “Rating” column is an ordinal variable that is treated as a categorical variable in this work, as we include the ordinal type within the general categorical type.

Inferring the type and values of categorical variables proves useful in many cases by eliminating the need for manual work. For example, it provides enhanced type inference capability for existing libraries such as the Pandas library, which attempts to infer the data type for each column of a dataset when parsing the data into a DataFrame object. Although the Pandas library provides an extensive set of functions to load, manipulate and save tabular datasets, it can detect only Boolean variables as categorical and treats non-Boolean categorical variables as either integers or strings (e.g., the Pandas.read_csv function would respectively label the “Class Name” and “Rating” columns given in Fig. 4.1 as object and int64). In such cases, the user needs to manually change the inferred types and then specify the categorical values for each categorical variable (referred to as categories in the Pandas library). Similarly, it enables automatic construction of data dictionaries in the well-known Attribute Relation File Format (ARFF), which has been used notably to describe OpenML datasets including the UCI datasets (see Fig. 4.2 for an example).

To the best of our knowledge, these issues are not addressed by any existing work in the literature, except Bot (proposed by Majoor and Vanschoren, 2018), OpenML and Weka which tackle type inference based on heuristics such as labeling a column as categorical when the number of unique values is lower than a threshold (see Chapter 2.2 for a detailed discussion). In this thesis, we use machine learning rather than

Class Name	Review Text	Age	Rating
Jackets	This beauty ...	37	5
Dresses	I'll start by ...	29	3
Pants	I took these ...	47	4
...



```

...
@ATTRIBUTE Class Name {Jackets, Dresses, ...}
@ATTRIBUTE Review Text STRING
@ATTRIBUTE Age NUMERIC
@ATTRIBUTE Rating {1,2,3,4,5}
...

```

Figure 4.2: A dataset and a section of the corresponding ARFF file.

heuristics to infer the type of a data column, and show that our probabilistic approach can be more flexible than hard-choices made with heuristics. By taking into account both the syntax of the entries in a data column and the features that can indicate whether the column represents a categorical variable or not, the proposed method can detect the general categorical type. Moreover, we define *inference of categorical values* as *the task of identifying the possible values a categorical variable can take on*. We address this task by adapting `ptype` which can robustly determine the possible values of a categorical variable by identifying missing data and anomalies in a data column. Therefore, the proposed method and `ptype` can be used together to infer the type and values of categorical variables. Our contributions are as follows:

- We propose a predictor that can identify the data type (categorical, date, float, integer and string) for each column of a dataset, taking into account the predictions of `ptype` and additional features (Section 4.2).
- We address the inference of categorical values by adapting `ptype` (Section 4.2).
- We show that our methods outperform the existing methods using a large number of datasets (Section 4.3).
- We investigate the use of meta-data for inferring the type and values of categorical variables (Section 4.3).

4.2 Methodology

In this section, we first introduce our probabilistic method `ptype-cat` (Sec. 4.2.1) and then describe the use of `ptype` to identify categorical values (Sec. 4.2.2). Lastly, we describe how we use meta-data for these tasks (Sec. 4.2.3).

4.2.1 Probabilistic Type Inference for Categorical Variables

Our goal here is to obtain the posterior probability distribution of column type over the categorical, date, float, integer and string types, which is achieved in two steps. Initially, assuming that a column of data $\mathbf{x} = \{x_i\}_{i=1}^N$ has been read in where each x_i denotes the characters in the i^{th} row and N is the number of rows in a data column, we calculate the posterior probability distribution $p(t|\mathbf{x})$ of column type t over the date, float, integer and string types by running a modified form of `ptype` that excludes the Boolean type. If a data column is labelled with the date or float type according to this posterior probability distribution, we assume that the posterior probability for the categorical type is zero and use the distribution as it is. Otherwise, if a data column is labelled with the integer or string type, we employ a separate binary classifier to determine the posterior probability for the categorical type, where the initial posterior probabilities for the four types are treated as features. The resulting method is called `ptype-cat`.

Note that we discard the Boolean feature used in the default setting of `ptype` as it leads to a limited capability to detect the categorical type. Instead, we propose four new features to characterize the categorical type. Two of our proposed features are the number of unique values in a data column and the uniqueness ratio, which are respectively denoted by U and R where R is defined as U/N . We extract the same features by taking into account the “clean” entries of a data column rather than all the data entries. Note that the clean entries refer to the data entries which are neither missing nor anomalous. These features are respectively denoted by U_c and R_c , where R_c is defined as U_c/N_c . Therefore, we obtain 8 features after combining `ptype` features with ours. These features are used by our separate binary classifier, which we describe next.

When a data column is labelled with the integer or string type, we determine the posterior probability for the categorical type by re-distributing the probability mass for the integer or string type according to a trained model, e.g., if the data type is initially inferred as integer, we divide the posterior probability for the integer type between the integer and categorical types according to a trained model as shown in Fig. 4.3.

To train the binary classifier for categorical/not-categorical classification, we use data columns annotated as integer, string and categorical. Mapping the integer and string labels to not-categorical, we train a binary Logistic Regression via 5-fold nested cross-validation, where we estimate its hyperparameters through grid-search. Note that

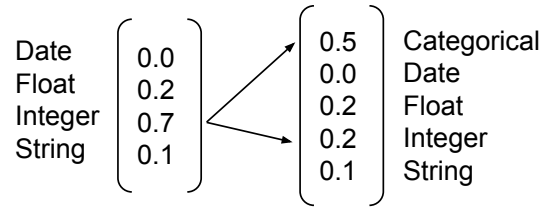


Figure 4.3: A graphical representation of the re-distribution step where we split the probability mass for the integer type between the integer and categorical types.

the hyperparameter of Logistic Regression and the corresponding range of values used in the grid-search are reported in Sec. 4.3.1.

We now introduce our notation for ptype-cat. In addition to the notation used for ptype, we let $\tilde{\mathbf{x}} = \{\tilde{x}_d\}_{d=1}^8$ denote the features extracted from \mathbf{x} . Note that we obtain 8 features by combining $p(t|\mathbf{x})$ with U , R , U_c and R_c . Additionally, we denote the final data type of \mathbf{x} by \tilde{y} . Therefore, the type inference task becomes a supervised learning problem where we train a separate classifier C using the set of $\{\tilde{\mathbf{x}}^j, \tilde{y}^j\}_{j=1}^M$. Here, j and M respectively denote the j^{th} data column and the total number of data columns.

4.2.2 Identification of Categorical Values

The task here is to infer the possible values a given categorical data column can take on. A naive approach would be to treat all the unique values in a column as the corresponding categorical values. However, this method, which we call Unique, would fail when the data contains missing and anomalous values, as such values would be labeled as valid categorical values rather than being discarded. To address this problem, we employ ptype which labels the entries of a data column that are not missing and anomalous as “clean”. We treat these clean entries as the categorical values of a data column. ptype can be used to calculate the corresponding posterior probability of row type being “clean” for each unique entry, which is denoted by $p(z = t | t = k, x_u)$ where x_u denotes the u^{th} unique data value and k denotes the k^{th} column type.

4.2.3 Using Meta-data for Inference

As an alternative approach to the methods described above, we have investigated whether the meta-data can help to address the inference tasks. We consider datasets for which meta-data are available, and extract a text sequence for each data column of a dataset. Below we describe how we use these text sequences for type inference and

identification of categorical values.

Using meta-data for type inference: We cast the type inference task as a sequence classification task where the goal is to label each text sequence with a type. For example, the meta-data of the Eucalyptus dataset contains the following sequence for the DBH column:

“12. DBH - best diameter base height (cm) - real”

Our goal here is to classify this sequence as float. Note that we could treat this problem as a tagging task and use a Named Entity Recognition (NER) approach; however, the data types are sometimes described by their synonyms or related terms (e.g., “real” is a synonym of the float type). Moreover, the data types may not be explicitly mentioned in the sequence. Therefore, type inference would be challenging for an NER based method. Instead, we employ the following sequence classification methods: Keyword Search, Bidirectional Long Short Term Memory (Graves and Schmidhuber, 2005) and RoBERTa (Liu et al., 2019), which are briefly described below:

- **Keyword Search (KS):** We construct a dictionary of type synonyms (e.g., binary is a synonym of the categorical type) from the training data (Table 4.1 presents a collection of synonyms found in our text sequences). Then we count the occurrences of synonyms per type in a test sequence, normalize these counts so that their sum is 1, and use these as the predicted scores of the KS method.
- **Bidirectional Long Short Term Memory (Bi-LSTM):** We use the well-known Bi-LSTM model for our text sequence classification task. Additionally, we combined Bi-LSTM with an attention mechanism to weight the contributions of each word to the classification decision. However, the results obtained by Bi-LSTM with an attention mechanism are omitted as they are not comparable with Bi-LSTM’s results.
- **RoBERTa:** We use the state-of-art method RoBERTa, which extends BERT (Devlin et al., 2019). BERT is a bi-directional transformer that outperforms Bi-LSTM and Bi-LSTM-Attention. We fine-tune the pre-trained RoBERTa model with our datasets for type inference.

Using meta-data for Identification of Categorical Values: The task of identifying categorical values can also be addressed using meta-data. Consider the sequence below that describes the Standard of Living Index column of the Contraceptive Method Choice dataset:

type	synonyms
binary	binary, boolean, boolean-valued,
categorical	categorical, discrete, enum, enumerated list, list/logical, logical, nominal, ordinal
date	date, date/time, time
float	continuous, float, number, numeric, numerical, numeric real, percentage, real
integer	int, integer, positive integer
string	char, character, string, text

Table 4.1: Data types and their synonyms found in our data dictionaries.

“8. Standard-of-living index (categorical) 1=low, 2, 3, 4=high”

Here, the goal is to identify the categorical values of 1, 2, 3 and 4 which are highlighted by the pink color. This task is more suitable for an NER-like approach rather than a sequence classification approach. We assume that patterns in these sequences can indicate the presence of categorical values, particularly the pattern where a categorical value precedes another one with a separator (such as comma) in between. Therefore, we adapt Bi-LSTM and RoBERTa to this task.

4.3 Experiments

In this section, we first describe our experimental setup (Sec. 4.3.1) and then present quantitative results on type inference and identification of categorical values (Sec. 4.3.2).

4.3.1 Experimental Setup

We describe the datasets, evaluation metrics and methods used in our experiments below:

Datasets

For type inference, we have used 86 datasets obtained from various sources such as Kaggle², OpenML³ (randomly selected through the API) and UCI⁴. We have annotated each dataset in terms of data types and categorical values by hand, based on the available meta-data and the unique values in each data column. These datasets are briefly described in Appendix B. The data files, their sources and our annotations can be accessed via <https://bit.ly/2Ra2Vu7>. The datasets are split using 5-fold nested cross-validation, which is summarized in Algorithm 1. Once the datasets are split, we collect their columns in the corresponding folds. Note that we split at the dataset level in order to avoid bias in test data. Our datasets contain a total number of 2989 columns (900 categorical, 49 date, 1462 float, 513 integer and 65 string). The counts of data types in each fold can be found in Appendix B. For the identification of categorical values task, we are interested in the 900 categorical data columns. Although 690 columns contain fewer than 6 unique categorical values, the number of categorical values in a column is between 1 and 80 (the data column with 80 categorical values, which is obtained from the CleanEHR dataset⁵, denotes the reason for a patient’s admission following the ICNARC Coding Method⁶).

Meta-data is available for 47 datasets out of 86. This leaves us with a total number of 897 text sequences (424 categorical, 33 date, 201 float, 214 integer and 25 string) extracted from meta-data. Additionally, we use all text sequences of categorical variables for identifying categorical values (these 424 sequences are obtained from 35 datasets out of 47). Note that all categorical values are explicitly mentioned in a total number of 183 sequences. The available sequences can be accessed via <https://bit.ly/2Ra2Vu7>.

Evaluation Metrics

We use different sets of metrics for type inference and identification of categorical values. For type inference, we use the metrics which are used to evaluate ptype in Chapter 3, namely overall accuracy and the Jaccard index. See Section 3.3.1 for a detailed de-

²<https://www.kaggle.com/datasets> [Accessed on 18/11/2020]

³<https://www.openml.org/search?type=data> [Accessed on 18/11/2020]

⁴<https://archive.ics.uci.edu/ml/datasets.php> [Accessed on 18/11/2020]

⁵The data is accessible via <https://github.com/ropensci/cleanEHR/tree/master/data> [Accessed on 04/12/2020].

⁶The details are available at <https://www.icnarc.org/Our-Audit/Audits/Cmp/Resources/Icm-Icnarc-Coding-Method> [Accessed on 04/12/2020].


```

Data      : X
Model    : M
Parameters: P, K
for  $i = 1$  to  $K$  do
  Split X into  $X_i^{training}$ ,  $X_i^{test}$ 
  for  $j = 1$  to  $K$  do
    Split  $X_i^{training}$  into  $X_{ij}^{training}$ ,  $X_{ij}^{test}$ 
    foreach  $p \in P$  do
      Train  $M$  on  $X_{ij}^{training}$ 
      Calculate Error  $E_{ijp}^{test}$ 
    end
  end
  Calculate Average Error  $E_{ip}^{test}$ 
  Select  $p^*$  where  $E_{ip}^{test}$  is minimum
  Train  $M^*$  on  $X_i^{training}$ 
  Calculate Error  $E_i^{test}$ 
end
Calculate Average Error  $E^{test}$ 

```

Algorithm 1: K-Fold Nested Cross-Validation

scription of these metrics and how they are used for type inference. Additionally, we plot the Precision Recall (PR) curve for each method and report the Average Precision (AP) of each curve. These curves are obtained by micro-averaging over folds, meaning that the output probabilities of a method are concatenated across five outer folds of the nested cross-validation, and across the samples in each fold.

For identification of categorical values, we first evaluate the methods using overall accuracy. The accuracy is 1 when the set of annotated categorical values is equal to the set of predicted categorical values and 0 otherwise. In order to take into account the partial matches between two sets, we calculate the Jaccard index per data column $J(A, B)$ defined as $|A \cap B| / |A \cup B|$, where A and B respectively denote the sets of annotated and predicted categorical values. Then we report their average over columns.

Methods

Type Inference: On type inference, we compare ptype-cat with Bot, OpenML and Weka. For Bot (Majoor and Vanschoren, 2018), we use the original implementation at

https://github.com/openml/ARFF-tools/blob/master/1030843_TheDataEncodingBot.ipynb. Note that by default Bot considers only a subsample of a data column for computational efficiency. Here, we feed all the entries into the method in order to eliminate any bias. Additionally, we treat the pre-defined threshold for the number of unique values, which is 100 by default, as a hyperparameter. We use two different hyperparameters for the integer and string types to allow a wider search space since the lower thresholds are respectively 10 and 25. We estimate these parameters via nested cross-validation using a grid-search over the intervals of $\{10, 20, \dots, 120\}$ and $\{25, 35, \dots, 125\}$.

For ptype-cat, we use Logistic Regression with an L2 penalty with the regularization strength parameter selected in the interval of $\{10^{-4}, \dots, 10^4\}$.

As we discuss in Chapter 2, the CSV to ARFF conversion methods used in OpenML and Weka are not directly applicable to our task. However, we adapt these methods by using ptype. We use ptype’s prediction when a data column is labelled with the ARFF label numeric to classify the column either as float or integer. For OpenML’s csv2arff method⁷, we use the original implementation at <https://github.com/openml/ARFF-tools/blob/master/csv-to-arff.py> and treat the number of unique values as a hyperparameter. Although its default value is 10, we estimate this parameter via nested cross-validation using a grid-search over the interval of $\{10, 20, \dots, 120\}$. For Weka’s method, we use the original implementation at https://waikato.github.io/weka-wiki/formats_and_processing/converting_csv_to_arff/ which does not have any hyperparameters.

Identification of Categorical Values: The methods adapted from OpenML and Weka can also be used for the identification of categorical values. Similarly, we adapt Bot to this task with a simple modification. Bot discards the data values that occur less than a threshold and does not treat them as categorical values. Here, we treat this threshold as a hyperparameter and estimate it via nested cross-validation using a grid-search over the interval of $\{5, 10, 20, \dots, 80\}$. In addition, we construct a baseline called Unique for inferring categorical values. Unique treats all unique values in a column as categorical values and is compared with ptype to demonstrate how much we can improve by eliminating missing and anomalous data.

Meta-data for Inference: We use Bi-LSTM and RoBERTa for inferring types and categorical values from meta-data. Additionally, we use KS for inferring types from

⁷OpenML provides this type inference methodology for dataset owners; however, Joaquin Vanschoren (pers. comm.) informed us that many datasets on OpenML have been manually annotated by dataset authors.

meta-data. These are described in Sec. 4.2.3.

We implement Bi-LSTM using Keras. We use 100-dimensional pre-trained word vectors from GloVe to initialize our word embeddings. Embeddings for words which are not included in GloVe are randomly initialized to the same vector where each entry is sampled uniformly from the interval of $[0,1)$ and re-trained. The dropout rate is set to 0.4. We use Adam for parameter optimization with a learning rate of 10^{-3} and a batch size of 32. The number of hidden nodes of an LSTM layer is treated as a hyperparameter and is estimated via the nested cross-validation with a grid-search over the values of $\{10, 20, 40, 80\}$.

For RoBERTa, we set the number of training epochs to 4, use batch size of 8 and estimate the learning rate with a grid-search over the values of $\{5 \cdot 10^{-5}, 4 \cdot 10^{-5}, 3 \cdot 10^{-5}, 2 \cdot 10^{-5}\}$. The remaining parameters are set to the default parameters⁸.

Combining data and meta-data: Finally, we investigate whether we can improve the performance by combining multiple classifiers that are trained on data and meta-data. The idea is to fuse information from these two different sources and check whether they can be complementary to each other, i.e., we ask whether we can rely on a meta-data based classifier when a data based classifier is misleading and vice versa. For example, ptype-cat and RoBERTa produce two separate probability distributions over the column type for each data column, based on its entries and the corresponding text sequence in its meta-data, respectively. We combine these probability distributions and use the resulting distribution for type inference. Note that we apply a similar step on identification of categorical values where we combine two probability values that denote whether a data value is a categorical value or not.

There are various strategies that can be used to combine classifiers such as the *sum* and *product* rules (see Kittler et al. (1998) for a detailed discussion and comparison of different combination strategies). Here, we calculate the weighted average of the probabilities of the data and meta-data based classifiers per type. The weights are chosen from the interval of $(0, 1)$ such that their sum is equal to 1. We select 100 pairs of weights and report the results obtained by the pair that leads to the highest overall accuracy on test data. This allows us to report the best performance improvement possible as we obtain the upper bound performance. We have also chosen the weights based on the performance on training data; however, this did not lead to any overall improvement over the performance of best performing single model. This issue indi-

⁸available at <https://github.com/ThilinaRajapakse/simpletransformers#default-settings> and <https://s3.amazonaws.com/models.huggingface.co/bert/roberta-base-config.json>.

cates a mismatch between the patterns in our training and test datasets, which may be resolved by extending the datasets used in our evaluations.

4.3.2 Experimental Results

Here, we present our experimental results for type inference and identification of categorical values (both using the data and meta-data).

Type Inference

Table 4.2 presents the performance of the methods in terms of overall accuracy and Jaccard index. These results indicate that ptype-cat consistently outperforms the competitor methods, except for the float type where it performs very similarly to Weka. These improvements are generally thanks to the flexibility of our probabilistic approach, i.e., we train a probabilistic model to learn the relationship between data features and types, whereas the others employ certain heuristics to identify types (see Chapter 2 for a detailed discussion). Note that we obtain the hyperparameters of Bot as 10 and 25 respectively for the integer and string types and OpenML’s unique values hyperparameter as 10 across test folds.

Fig. 4.4 presents the PR curves obtained by the methods, which indicates a similar trend as above in that ptype-cat performs better than the other methods. Note that the competitor methods provide only either 0 or 1 as a score for each data type. In contrast, ptype-cat generates more fine-grained scores valued between 0 and 1.

	Method			
	Bot	OpenML	Weka	ptype-cat
Overall Accuracy	0.84	0.88	0.79	0.93
Categorical	0.68	0.81	0.43	0.85
Date	0.08	0.00	0.00	0.51
Float	0.83	0.95	0.97	0.97
Integer	0.64	0.59	0.49	0.70
String	0.29	0.20	0.06	0.52

Table 4.2: Performance of the methods using the overall accuracy and per-class Jaccard index, for the Categorical, Date, Float, Integer and String types.

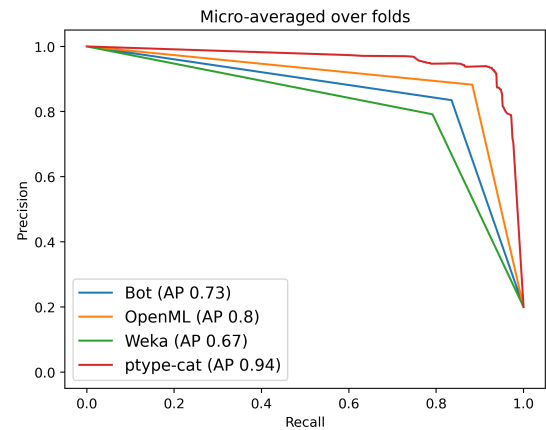


Figure 4.4: PR curves for the methods.

Fig. 4.5 presents the normalized confusion matrices for the methods, normalized so that each column sums to 1. All methods lead to confusions by classifying columns

as categorical rather than as integer or string. These failures are not surprising to some extent since categorical values are either encoded by integers or strings. However, ptype-cat has fewer such confusions than the others.

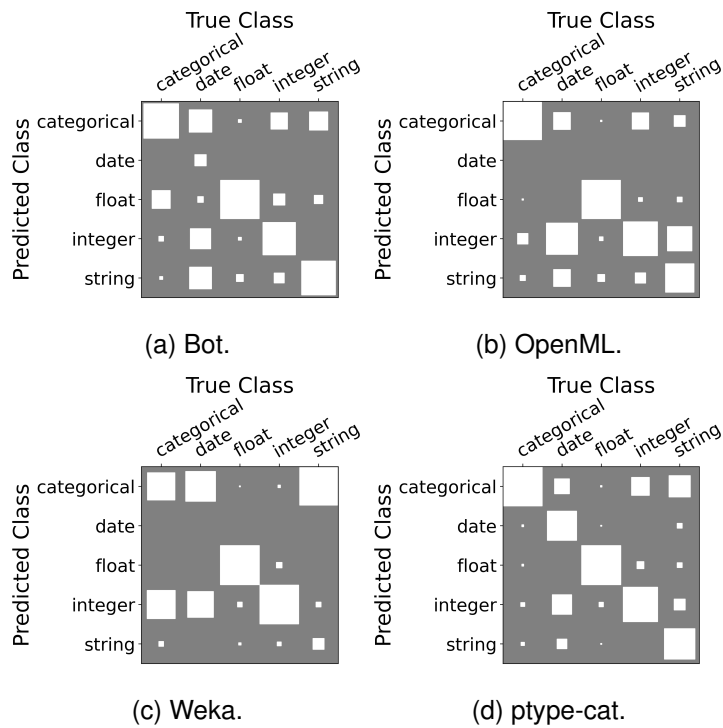


Figure 4.5: Hinton plots of the normalized confusion matrices.

ptype-cat performs better than the competitor methods for the date type, which can be explained by several reasons. The main reason for Bot is that it does not support date formats with time information. Instead, data columns in such formats are treated either as categorical or string, depending on the number of unique values. Additionally, Bot does not consider textual dates such as months and 4-digit formatted years, and results in misclassifications of categorical since the number of unique values is typically low. Weka considers the time information; however, it supports only the ISO-8601 format of “yyyy-MM-ddTHH:mm:ss”, unless the user specifies differently. Our method supports a more extensive set of formats including certain non-standard date formats thanks to the features obtained from ptype. OpenML completely discards the date type.

Compared to OpenML, our method correctly classifies an additional 141 data columns (62 categorical, 27 date, 39 float, 8 integer and 5 string). The main difference is that ptype-cat correctly classifies 51 categorical columns which are misclassified as integer by OpenML. There are two main reasons to explain this difference. First, OpenML predicts the data type as integer when the number of unique values is higher than 10

(e.g., most columns of the Poker Hand dataset contain 13 categorical values encoded by integers). Secondly, OpenML does not properly handle missing data when the categorical values are encoded by strings. To correctly label such columns, it requires all the entries to be converted to string by using the Python's *isinstance* function which fails when the Pandas library detects missing data and encodes them as `np.nan`. For example, OpenML initially labels the “HCMEST” column of the cleanEHR dataset (which contains values Y, N and NULL) as numeric rather than categorical. A forced choice between integer and float in `ptype` then results in equal posterior probabilities for these two types, as `ptype` must set its row latent variables to “anomaly” to explain the non-NULL data.

To determine whether the column type predictions of `ptype` and OpenML are significantly different, we apply the McNemar's test (see e.g., Dietterich 1998), which assumes that the two methods should have the same error rate under the null hypothesis. We compute the test statistic $(|n_{01} - n_{10}|)^2 / (n_{01} + n_{10})$, where n_{01} and n_{10} denote the number of test columns misclassified by only OpenML, and by only `ptype` respectively. In our case, n_{01} and n_{10} are respectively equal to 185 and 44, which results in a statistic of 85.6. If the null hypothesis is correct, then the probability that this statistic is greater than 3.84 is less than 0.05 (Dietterich, 1998). Thus this result provides evidence to reject the null hypothesis and confirms that the methods are statistically significantly different from each other.

Unlike the other methods, Bot misclassifies a high number of categorical variables as floats. These failures occur when the Pandas library fails to parse a given dataset correctly. For example, the “Active Sport” column of the Young People Survey dataset—a categorical variable which ranks how active a young person is from 1 to 5—is labelled with the float type by the Pandas library and consequently all integers are converted to their floating-point representations (e.g., 1.0 to 5.0). Therefore, Bot is fed with these floating-point numbers rather than integers resulting in confusions.

A common pattern in failure cases of all methods is that the assumption about the number of unique values does not always hold, i.e., there can be data columns of type integer or string with a low number of unique values. Consider the “State” column of the Geoplaces dataset which contains data values such as Morelos, S.L.P. and San Luis Potosi. The data column contains only 13 unique values out of 130 entries, which causes the methods to misclassify it as categorical (note that this data column is assumed to be of type string rather than categorical since the data is collected as free text). OpenML handles such cases slightly better than the others based on the heuristics

used. However, its overall performance for the string type is poor as it classifies quite a high number of columns as integer rather than string due to the presence of missing data.

Finally, we inspect the coefficients of the Logistic Regression classifiers to understand which features contribute to the categorical/non-categorical decision the most. We observe that the coefficients of U_c have the highest values. This is respectively followed by U and R . These observations indicate that the classifiers use our proposed features to detect the categorical type. Note that there is a strong positive correlation between the coefficients of U and U_c , i.e., the corresponding Pearson’s correlation coefficient is 0.99. On the other hand, we observe a less notable correlation between the coefficients of U and R with the Pearson’s correlation coefficient of 0.17.

Identification of Categorical Values

Table 4.3 presents the performance of the methods on identification of categorical values. These results indicate that ptype-cat outperforms the competitor methods in terms of both metrics, whereas the leading competitor method is OpenML. We also observe that Unique performs better than Bot and WEKA, which produce similar results. Note that we obtain the hyperparameters of Bot as 5 across test folds.

	Method				
	Bot	OpenML	Weka	Unique	ptype-cat
Overall Accuracy	0.33	0.83	0.38	0.64	0.90
Average Jaccard	0.40	0.87	0.42	0.87	0.92

Table 4.3: Performance of the models on inference of categorical values.

The accuracies indicate that OpenML identifies all categorical values correctly for a higher number of data columns than Unique. The difference in their overall accuracies results from the inability of Unique to detect missing data. For example, the “Chemox” column of the CleanEHR dataset has three unique values: 0, 1 and NULL. Here, the annotated categorical values are 0 and 1, and NULL encodes missing data. While OpenML correctly labels 0 and 1 as the categorical values, Unique treats NULL as another categorical value. On the other hand, the Average Jaccard score denotes that they provide similar coverage of categorical values per column. The main reason is that only a few data values are misclassified per column by Unique, which does not lead to large gaps in their performances.

Bot obtains the worst performance, which is not surprising as it relies on the number of occurrences of data values for detecting categorical values. Using the number of occurrences can become misleading for Bot. For example, it treats missing or anomalous data observed more than a pre-defined threshold in the data as categorical values. Similarly, it cannot identify categorical values that occur less than the same threshold.

Weka performs slightly better than Bot; however, its performance is still poor compared to the remaining methods. This is mainly because when it misclassifies a categorical variable, it generates an empty list for the corresponding categorical values, which causes both the accuracy and the Jaccard index to be zero.

In addition to the Average Jaccard score, we test whether methods produce statistically different Jaccard indices per column. We apply a paired t-test on the list of Jaccard indices obtained by Unique and ptype-cat (in the same order). Similarly, we obtain the p-values by OpenML-ptype-cat, Bot-ptype-cat and Weka-ptype-cat comparisons. We find that all the p-values are lower than 0.001. These results reject the null hypothesis that the means are equal and confirm that they are significantly different.

Meta-data for Inference

Meta-data for Type Inference: Table 4.4 presents the performance of KS, Bi-LSTM, RoBERTa and ptype-cat on type inference. Additionally, we evaluate RoBERTa + ptype-cat which is the method obtained by combining the predictions of RoBERTa and ptype-cat. Note that the first three approaches rely on the meta-data for type inference, whereas ptype-cat relies on the corresponding column data, i.e., we re-evaluate ptype-cat on the data columns for which text sequences are extracted from meta-data. However, RoBERTa + ptype-cat uses the information both in the data and meta-data. As per the table, RoBERTa leads to a better overall performance than KS and Bi-LSTM; however, ptype-cat consistently outperforms these three competitor methods. These results indicate that ptype-cat allows us to infer types accurately from data without looking at meta-data. Moreover, the results show that the performance obtained by ptype-cat can be slightly improved overall by combining its predictions with RoBERTa's predictions. Note that the performance of RoBERTa + ptype-cat denotes the best possible performance that can be obtained by combining these classifiers.

Fig. 4.6 presents the PR curves obtained by the methods, which indicates a similar trend as above in that combining RoBERTa and ptype-cat performs better than the other methods.

Fig. 4.7 presents the normalized confusion matrices for the methods, normalized

	Method				
	KS	Bi-LSTM	RoBERTa	ptype-cat	RoBERTa + ptype-cat
Overall Accuracy	0.50	0.62	0.68	0.90	0.92
Categorical	0.40	0.58	0.71	0.86	0.89
Date	0.42	0.11	0.11	0.59	0.59
Float	0.53	0.35	0.39	0.92	0.92
Integer	0.23	0.42	0.42	0.73	0.79
String	0.12	0.08	0.04	0.27	0.23

Table 4.4: Performance of the methods using the overall accuracy and per-class Jaccard index, for the Categorical, Date, Float, Integer and String types. We highlight the best score in each row by making the highest score bold.

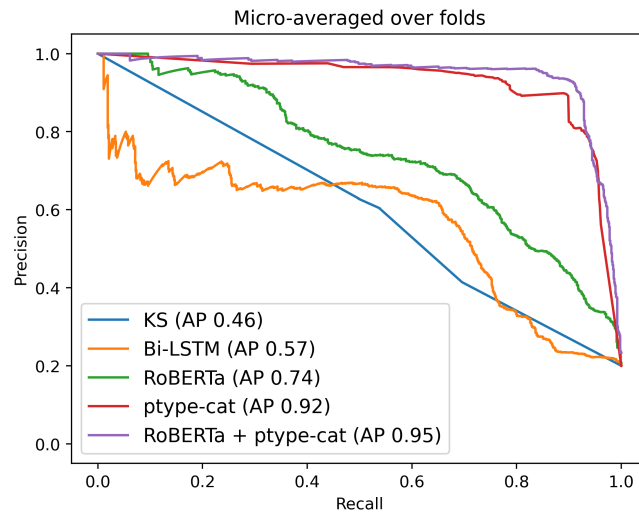


Figure 4.6: PR curves for the methods.

so that each column sums to 1. Below we discuss the common patterns in these normalized confusion matrices.

All methods lead to confusions by misclassifying numerical columns, e.g., classifying a column as float rather than integer. These failures are understandable to some extent since the corresponding text sequences are likely to share a similar context. However, ptype-cat has fewer such confusions than the others, as it uses the data values themselves rather than the text sequences. For example, the “IOBlank” column of the JM1 dataset⁹ is described by the “15. IOBlank : numeric % Halstead’s count of

⁹The dataset is collected for software defect prediction and is available at <http://promise.site.uottawa.ca/SERepository/datasets-page.html> [accessed on 04/03/2021]

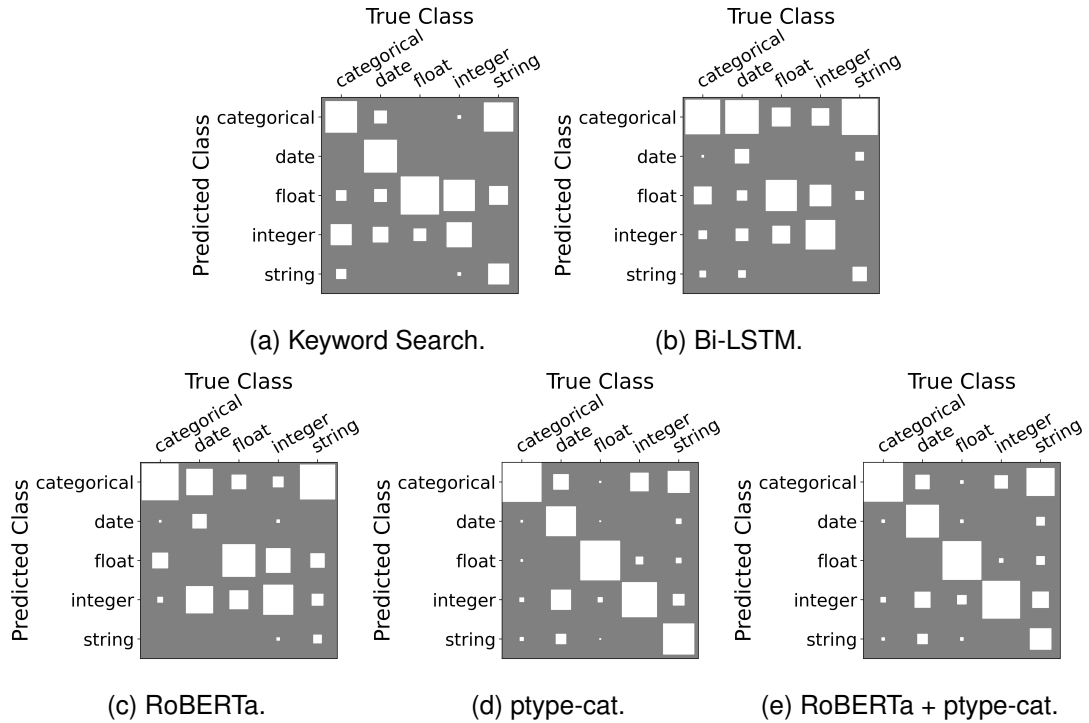


Figure 4.7: Hinton plots of the normalized confusion matrices.

blank lines” sequence in the meta-data. Here, only ptype-cat correctly identifies the type as integer, whereas the competitor methods classify the sequence as float. Similarly, the “Duration” column of the Bank Marketing dataset is described by the “12 - duration: last contact duration, in seconds (numeric)” sequence in the meta-data. Here, KS, Bi-LSTM and RoBERTa classify this sequence as float, whereas ptype-cat correctly identifies its type as integer.

Fig. 4.7 indicates that ptype-cat can identify the categorical type better than the competitor methods. Consider the “Study Time”¹⁰ column of the Student Alcohol Consumption dataset and the corresponding sequence of “studytime - weekly study time (numeric: 1 - 10 hours)”. Here, ptype-cat can correctly detect the type based on the features extracted from the data column itself (the column consists of 395 entries with 4 unique values). However, all the competitor methods fail, i.e., Bi-LSTM labels this sequence with the float type, whereas RoBERTa label it with the integer type and KS cannot assign any type. Note that KS fails because there is no explicit mention of any type in this sequence that it is familiar with, including “numeric” which is not observed in the training set for the corresponding nested cross-validation fold.

¹⁰We assume that the column denotes an ordinal variable as it consists of the unique values of 1, 2, 3 and 4 which have a natural order among themselves. Note that the ordinal type is included within the general categorical type in this work.

Although Bi-LSTM and RoBERTa do not rely on the presence of explicit mentions, the patterns in the sequence are perhaps too rare in the training data or too complex for the methods to capture enough information about the categorical type.

We observe that the date and string types are challenging to identify by using the meta-data, which can be explained by several reasons. First, the syntax of the values in a data column may be a more direct indicator of its type than the textual descriptions of a column. Secondly, the date and string types are rare classes in our datasets. Therefore, there may not be enough examples for the training to be effective for these types.

Lastly, we compare the predictions of ptype-cat and RoBERTa + ptype-cat. We obtain RoBERTa + ptype-cat by using the weights of 0.33 and 0.67 for RoBERTa and ptype-cat, respectively. We observe that RoBERTa + ptype-cat identifies the types of an additional number of 19 columns (11 categorical and 8 integer). However, it also misclassifies one more column of type string than ptype-cat. A detailed inspection of their predictions shows that ptype-cat treats several columns as integer rather than categorical with low confidence. For example, the “Course_instructor” column of the TAE dataset, which denotes the IDs of course instructors, consists of 151 entries where there are 25 unique categorical values. ptype-cat labels this column with the integer type because the corresponding posterior probability values for the categorical and integer types are respectively 0.34 and 0.66. However, RoBERTa is confident that the column is of type categorical with a posterior probability value of 0.97. Therefore, we obtain the correct label, which is categorical, by combining these two models (weighted averaging the prediction scores let us obtain the posterior probability values of 0.55 and 0.45 for the categorical and integer types, respectively).

Meta-data for Identification of Categorical Values: Table 4.5 presents the performance of Bi-LSTM, RoBERTa and ptype-cat on identification of categorical values. Note that the first two approaches rely only on the meta-data for inference, whereas ptype-cat relies on the corresponding data, i.e., we re-evaluate ptype-cat on the data columns for which text sequences are extracted from meta-data. On the other hand, RoBERTa + ptype-cat uses both the data and meta-data, where the weights of RoBERTa and ptype-cat are respectively found to be 0.01 and 0.99 via a grid-search as discussed in Sec. 4.3.1. As per the table, RoBERTa leads to a better overall performance than Bi-LSTM; however, ptype-cat consistently outperforms the competitor methods. These results indicate that ptype-cat allows us to infer categorical values accurately from data without looking at meta-data.

	Method			
	Bi-LSTM	RoBERTa	ptype-cat	RoBERTa + ptype-cat
Overall Accuracy	0.04	0.27	0.78	0.78
Average Jaccard	0.10	0.35	0.81	0.81

Table 4.5: Performance of the methods using Overall Accuracy and average of Jaccard index per column.

Figure 4.8 denotes the PR curves obtained by the methods, which indicate a similar pattern as above in that ptype-cat performs better than Bi-LSTM and RoBERTa. However, we now observe slight improvement in the AUC by combining RoBERTa and ptype-cat. This slightly improvement shows that we are slightly more confident about our predictions, although the predicted labels stay the same. Note that the performance of RoBERTa + ptype-cat corresponds to the best possible performance that can be obtained by combining these models.

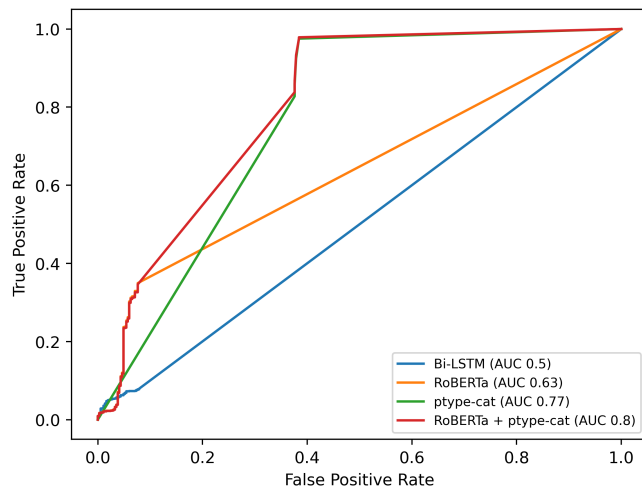


Figure 4.8: PR curves for the methods.

Note that our goal here is to identify the categorical values observed in the data. However, Bi-LSTM and RoBERTa cannot identify the categorical values that are not explicitly mentioned in meta-data. Therefore, the poor performances obtained by using Bi-LSTM and RoBERTa may not be surprising as only 183 sequences out of 424 contain all categorical values. For example, for the “protocol_type” column of the KDDCup99 dataset, we are given the sequence of “protocol_type type of the protocol, e.g., tcp, udp, etc. discrete”. Here, Bi-LSTM does not identify any categorical values, whereas RoBERTa correctly labels tcp and udp as categorical values. However, the

data column contains an additional categorical value of `icmp`. `ptype-cat` is the only method that could identify all three categorical values in this example.

4.4 Discussion

Above, we address the data type problem which tackles the classification of a data column into categorical, date, float, integer or string. Our method takes into account both the syntax of the data entries and the additional features that can indicate whether a data column is of categorical type or not. Additionally, we adapt `ptype` to the task of identifying the possible values a categorical variable can take on. We demonstrate improved accuracy for these tasks over competitor methods, which rely on heuristics. Moreover, we investigate the use of meta-data for inferring the type and values of categorical variables and show that, although relying on meta-data only does not perform as well as our model, it can help to slightly improve the performance of our model for type inference when combined.

As a next step, we plan to improve the impact of our work in practice by producing a software tool based on `ptype-cat`, that can be used by the practitioners. In this regard, we are going to train a final version of the proposed model by using all the available datasets and incorporate this trained model into the Python package of `ptype`. Note that the `ptype` package already supports a number of user interactions as we discuss in Chapter 3.3.3; however, `ptype-cat` may bring additional challenges, such as the design of user-interactions where the main goal is to let the user modify the incorrect annotations of our model for type and categorical values.

Finally, it may be useful to distinguish ordinal and nominal variables in certain scenarios rather than treating them as categoricals as in `ptype-cat`. This can be accomplished by feeding `ptype-cat` with new features extracted from categorical values themselves. For example, Hernández-Lobato et al. (2014) discriminate between the ordinal and nominal types by comparing the model evidence and the predictive test log-likelihood of ordinal regression models and multi-class classifiers. One could run such models on data and use their outputs as features for `ptype-cat`. It can be possible to obtain similar features by using on word-embeddings for categorical values encoded as strings.

Chapter 5

Bringing Semantics into Type Inference

This chapter begins by describing our motivation to use semantics for type inference (Sec. 5.1). Then, we describe the proposed methodology (Sec. 5.2) and discuss the related work (Sec. 5.3). Finally, we present our experiments (Sec. 5.4), which is followed by a summary of this work and a discussion of potential research directions (Sec. 5.5).

5.1 Introduction

In Chapters 3 and 4, we describe two type inference methods named `ptype` and `ptype-cat`. `ptype` relies on the syntax of the data values for type inference, whereas `ptype-cat`, in addition to the syntax, takes into account additional features to better detect the categorical type. However, they do not make use of the semantic information about a given dataset, which can be essential for accurate and enhanced type inference.

Consider the following example: Suppose we are given the quantitative data column¹ in Figure 5.1(a), which denotes the volume of freezers in various households. The value of each data entry is a *measurement* encoded by a unit symbol, except the last entry which does not have any unit symbol (we refer to the absence of a unit symbol in an entry as the *missing unit*). Note that a measurement is a combination of a numeric value and a unit (e.g., litre), which can be encoded by several possible unit symbols for that unit (e.g., `l` and `L`). As shown in Figure 5.1(b), the measurements are encoded with two distinct units (litres and cubic feet) and six different unit

¹The data is sampled from the “Freezer_volume” column of the Household Electricity Survey (HES) dataset which can be accessed by registering at <https://tinyurl.com/ybbqu3n3>.

symbols, some of which including `ltrs`, and `Cu` can be considered as *anomalous unit symbols* as they do not follow the standard encodings of units. Moreover, a unit (e.g., litre) can be informative about the dimension of a measurement (in this case volume). The International Vocabulary of Metrology² defines dimension as an expression of the dependence of a physical quantity on mutually independent components called base quantities. Following the terminology used by Chambers and Erwig (2010), we categorize dimensions into two groups: (i) basic dimensions (such as length and time) and (ii) derived dimensions (such as speed and force).

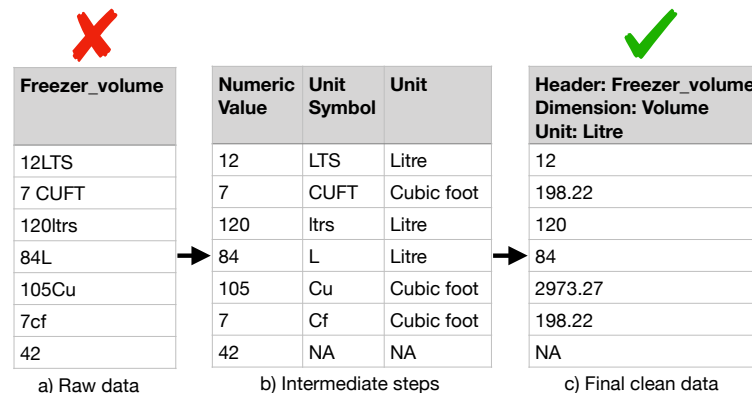


Figure 5.1: A motivating real-world example that represents our pipeline. a) shows the samples of a raw dataset. b) indicates the intermediate steps required to transform the column. c) denotes the final data column obtained by applying the transformations.

`ptype` would label the column in Figure 5.1(a) with the string type rather than a numeric type (float or integer) as the data values consist of alphanumeric characters and whitespace. Similarly, `ptype-cat` would treat it as either string or categorical depending on the additional features (e.g., the number of unique values) rather a numeric type. But the correct type would be float if all the measurements are encoded in litres, as in Figure 5.1(c). Moreover, we may want to extract additional semantic information about the column such as its dimension, which is volume, and the units of all its entries so that the data entries can be canonicalized (i.e., so they all are expressed in the same units). Such information cannot be obtained by using `ptype` or `ptype-cat`. Instead, one needs to apply transformations such as parsing and identifying the units in the entries, inferring the common unit for the column which is placed in the metadata such as the header, making the entries numeric and scaling the entries where needed. The

²The document is accessible at https://www.bipm.org/documents/20126/2071204/JCGM_200_2012.pdf/f0e1ad45-d337-bbeb-53a6-15fe649d0ff1

resulting data column would then be easier to understand and can be directly used in data analytics pipelines.

Samadian et al. (2014) argue that clinical data often suffer from these problems, as the data is usually collected by uncoordinated groups of people. Such issues are often manually identified and resolved with dataset-specific scripts, which are typically used only once. However, they should be automated with a dataset-independent tool to reduce the time and effort spent on manual transformation of the data, and to enhance reproducibility. To the best of our knowledge, the tasks below are not addressed by any existing work in the literature (see Chapter 5.3 for a detailed discussion):

- the task of inferring the dimension of a column,
- the task of identifying the unit for each entry of a column,
- the task of canonicalizing the entries of a column.

In this chapter, we propose *p*type-*s*emantics - an extension of *p*type enriched by incorporating semantic information about units of measurement. The proposed model allows us to extract semantic information about a given data column (such as its dimension and unit), canonicalize its entries and provides enhanced type inference capabilities for syntactic type inference methods such as *p*type. Our contributions are as follows:

- We propose a probabilistic model which annotates data columns containing unit symbols in terms of dimensions and units (Section 2).
- We make the first quantitative comparison of the existing methods on the unit identification task in real-world tabular data (Section 4).
- We present the first set of real-world datasets annotated for the units of measurement, to accelerate research in this area.

5.2 Methodology

This section describes how we represent units (Sec. 5.2.1), introduces our model (Sec. 5.2.2) and presents the inference in this model (Sec. 5.2.3).

5.2.1 Representing Units

We represent units by extending a dictionary of units curated from Wikipedia³ with information from WikiData (Vrandečić and Krötzsch, 2014) and QUDT (Quantities, Units, Dimensions and Data Types Ontology)⁴. Keil and Schindler (2018) show that WikiData is the most comprehensive knowledge graph for units and that QUDT contains additional information to WikiData. By extending the existing dictionary, we increase the number of units from 284 to 1080 (we only consider the units in English, although extensions to other languages are straightforward).

Table 5.1 presents two instances that respectively represent the units of litre and gram. The instances in the original dictionary can have six attributes: name, surfaces, entity, URI, dimensions, and symbols. Note that the “surfaces” attribute denotes a list of *strings* that refer to a unit, whereas the “symbols” attribute is a list of possible symbols and abbreviations for that unit.

name	surfaces	entity	URI	dimensions	symbols
litre	cubic decimetre, litre, cubic decimeter, liter	volume	.../wiki/Litre	{'base': 'decimetre', 'power': 3}	l, L, ltr
gram	gram, gramme	mass	.../wiki/Gram	—	g, gm

Table 5.1: Two elements of the unit dictionary. Note that URIs begin with `https://en.wikipedia.org/`.

For each instance we extract a dimension, a unit and a list of unit symbols. The name and entity attributes of the instances in Table 5.1 are respectively used as units and their dimensions. Note that the naming convention used in the dictionary differs from our terminology in that it refers to dimension as entity and uses the “dimensions” attribute to encode the relationship between units. The “surfaces” and “symbols” attributes are combined to build a set of unit symbols. To search for additional symbols of a unit, we query WikiData using the URI attribute. Additionally, the QUDT reference ID, when available in the response obtained from WikiData, is used to query QUDT. We provide the details of this process in Appendix C.1 for reproducibility. The resulting set of tuples are then used to construct our model and can be seen as training data.

³Accessible at <https://github.com/marcolagi/quantulum/blob/master/quantulum/units.json>

⁴<https://www.qudt.org/>

5.2.2 The Proposed Model

To make use of the semantic information in the entries of a data column that contain unit symbols, we extend ptype by incorporating the knowledge of units from knowledge graphs. The resulting model called ptype-semantics can generate more fine-grained type predictions than the string type, namely the column’s dimension (such as length, mass or volume). In addition, it can detect the anomalous entries with non-standard unit symbols which would be treated as valid string typed entries by ptype, and can automatically repair anomalous unit symbols by mapping them to known unit symbols.

ptype-semantics first generates a column dimension from a set of possible dimensions and then generates a row unit for each data entry from the set of possible units for that column dimension. Next, the model generates row labels, each of which is either equal to the corresponding row unit, missing or anomalous. Finally, the observation model generates a unit symbol for each row according to its label. We now introduce our notation to represent this process, which is summarised in Table 5.2

Symbol	Description
t	the column dimension
u_i	the unit of the i^{th} row
z_i	the label of the i^{th} row
v_i	the numeric value of the i^{th} row
x_i	the unit symbol of the i^{th} row
y_i	the characters of the i^{th} row
K	the number of possible dimensions
L_t	the number of possible units for dimension t
S_{u_i}	the number of possible unit symbols for unit u_i

Table 5.2: A summary of the notation used by ptype-semantics.

We assume that a column of data $\mathbf{y} = \{y_i\}_{i=1}^N$ consisting of N rows has been read in, where each y_i denotes the characters in the i^{th} row. Additionally, each y_i is assumed to be parsed to a numeric value v_i and a unit symbol x_i , which may be missing for some entries, i.e., x_i may be null. In this work, we use regular expressions to parse observations \mathbf{y} (see Sec. 5.2.2 and Appendix C.3 for the details). We propose a generative model with a set of latent variables t , $\mathbf{u} = \{u_i\}_{i=1}^N$ and $\mathbf{z} = \{z_i\}_{i=1}^N$, where t denotes the dimension of a column, u_i the unit and z_i the label of its i^{th} row. The missing and anomalous labels, denoted by m and a respectively, are used to model the data entries

where the unit symbols are missing or anomalous. Thus, each z_i can be m or a as well as a row unit that fit the column dimension, i.e. $z_i \in \{\text{Litre, Cubic foot, \dots, } m, a\}$ given that t is volume. With this noise model, we make our inference procedure robust against missing and anomalous unit symbols.

Denoting the number of possible dimensions for a column by K , our model has the following generative process:

$$\begin{aligned} \text{column dimension } t &\sim \mathcal{U}(1, K), \\ \text{row unit } u_i &\sim p(u_i|t), \\ \text{row label } z_i &= \begin{cases} u_i & \text{with probability } w_{u_i}^{u_i}, \\ m & \text{with probability } w_{u_i}^m, \\ a & \text{with probability } w_{u_i}^a, \end{cases} \\ \text{row symbol } x_i &\sim p(x_i|z_i), \end{aligned}$$

where L_t , W and $p(x_i|z_i)$ are respectively the number of units for the dimension t , the mixing proportions, and the observation model. \mathcal{U} denotes a discrete Uniform distribution. Additionally, $p(u_i|t)$ denotes how likely a row unit u_i represents the dimension t . Here the mixing proportions $w_{u_i}^{u_i} + w_{u_i}^m + w_{u_i}^a = 1$ for each row unit u_i , and $p(u_i|t)$ is modeled with an indicator function that assigns a non-zero score only when a unit is a known unit of a dimension. Since entries are often expected to be of a regular row label rather than the missing or anomalous labels, we favour regular labels during inference by using lower coefficients for the missing and anomalous labels, i.e. $w_{u_i}^m < w_{u_i}^{u_i}$ and $w_{u_i}^a < w_{u_i}^{u_i}$. These mixing proportions W are assumed to be fixed and known. Even though one could also learn the mixing proportions, this may not be vital as long as the coefficients of the regular labels are larger than the others.

We build the observation model $p(x_i|z_i)$ upon three functions. First, we develop a Categorical distribution for row unit u_i where the categories correspond to the possible unit symbols for that unit:

$$p(x_i|z_i = u_i) = \text{Cat}(x_i, \pi_{u_i}), \quad (5.1)$$

where $\sum_{s=1}^{S_{u_i}} \pi_{u_i}^s = 1$. Here, S_{u_i} denotes the number of known unit symbols for a unit u_i . Second, we model missing units with an indicator function, which assigns a non-zero probability only when a unit symbol is missing. Lastly, we adapt the anomaly type in ptype, which is built based on the idea of an *X-factor* proposed by Quinn et al. (2009), to model anomalous unit symbols. Here, we introduce a likelihood function

that assigns low non-zero probabilities to any data value, which in turn allows the model to detect anomalous unit symbols which do not fit any known unit.

Parsing Unit Symbols

Unit symbols are usually positioned after quantities as in 1 L, with some exceptions where the conventions are different. For example, they are usually placed before quantities to represent monetary amounts, e.g., \$159000 and \$85810. When abbreviations are used, however, unit symbols are placed after numeric parts, e.g., 70 USD, 19.68 AUD. Monetary amounts can also be represented in various non-standard formats. For example, whitespace may be placed between symbols and amounts, e.g., \$ 1012, \$ 964. We develop regular expressions by taking into account possible positions of unit symbols. Additionally, we remove leading and trailing whitespace as well as trailing dots. The regular expressions used can be found in Appendix C.3 for reproducibility.

5.2.3 Inference

Given the row symbols \mathbf{x} in a data column, the initial task is to infer the column dimension t , which is cast as the problem of calculating the posterior distribution of t given \mathbf{x} , namely $p(t|\mathbf{x})$. We then compute a posterior distribution over each row label conditioned on the dimension and the observed value, i.e., $p(z_i|t, x_i)$. Next, we determine the row units by calculating the posterior distribution of each row unit u_i given t , z_i and x_i , which is also used to predict the column unit. Lastly, we introduce a strategy to map a unit symbol labelled as anomalous to a known unit symbol and briefly describe how we canonicalize the units. Note that we define *unit canonicalization* as the task of canonicalizing the entries of a data column so that they all are expressed in the same units. The detailed derivations are presented for reproducibility in Appendix C.4; here we briefly discuss the corresponding calculations.

Column Dimension Inference

Assuming that the entries of a data column are conditionally independent given the column dimension, we obtain the posterior distribution of column dimension t by

marginalizing over row unit and label variables \mathbf{u} and \mathbf{z} as follows:

$$p(t = k | \mathbf{x}) \propto p(t = k) \prod_{i=1}^N \left[\sum_{l=1}^{L_t} p(u_i = l | t = k) \left(w_l^l p(x_i | z_i = l) + w_l^m p(x_i | z_i = m) + w_l^a p(x_i | z_i = a) \right) \right]. \quad (5.2)$$

Eq. 5.2 can be used to estimate the column dimension t , since the one with maximum posterior probability is the most likely dimension corresponding to the column \mathbf{x} . Note that Eq. 5.2 is similar to Eq. 3.1 as column dimension inference and column type inference are carried out similarly (see Section 3.2.3 for the details).

As per equation 5.2, the model estimates the column dimension by considering all the data rows, i.e. having missing or anomalous unit symbols does not confuse the dimension inference. Note that such entries would have similar likelihoods for each column dimension, which allows the model to choose the dominant dimension for regular entries.

Row Label Inference

Following the inference of column dimension, the posterior probabilities of each row label z_i given $t = k$ and x_i is obtained by marginalizing the latent unit variable u_i as follows:

$$p(z_i = j | t = k, x_i) \propto \sum_{l=1}^{L_k} p(u_i = l | t = k) w_l^j p(x_i | z_i = j). \quad (5.3)$$

Note that Eq. 5.3 is similar to Eq. 3.2 as row label inference and row type inference are carried out similarly (see Section 3.2.3 for the details).

Row Unit Inference

Given $t = k$, $z_i = j$ and x_i , the posterior distribution of row unit u_i is obtained as:

$$p(u_i = l | t = k, z_i = j, x_i) = \frac{p(u_i = l | t = k) p(x_i | z_i = j)}{\sum_{u_i=1}^{L_k} p(u_i = l | t = k) p(x_i | z_i = j)}. \quad (5.4)$$

Column Unit Inference

Following the column dimension inference, we set the column unit l^* as follows:

$$l^* = \operatorname{argmax}_l \sum_{i=1}^N p(u_i = l | t = k, x_i), \quad (5.5)$$

where $l \in \{1, \dots, L_t\}$ denotes a possible unit for dimension t .

Correcting Anomalous Unit Symbols

Row label inference annotates each data entry either as a unit, missing or anomalous. We assume that the units of anomalous entries are encoded by anomalous unit symbols (e.g., `ltrs` for `litres`) and can be identified by mapping anomalous unit symbols (e.g., `ltrs`) to known unit symbols (e.g., `lt`) based on the edit-distance (Levenshtein, 1966). The edit-distance measures the minimum number of operations (addition, deletion or substitution) that needs to be done to transform a string to another, and can handle misspellings and non-standard abbreviations. Note that we restrict the set of unit symbols to be compared with according to the column dimension, i.e., we compare `ltrs` with known unit symbols that encode units of volume when the column dimension is inferred as volume.

Canonicalizing Units

Following the inference of the row units and the column unit, we are now interested in representing each row with the same unit by scaling its numerical value (e.g., converting the data entry `1 m` to `100 cm` when the column and row units are respectively centimetres and metres). Currently, we convert units via an existing tool named `Pint` (Grecco, 2019) and demonstrate that our model improves over its performance for unit conversion by identifying the row units more accurately. See Sec. 5.3 for a detailed discussion of `Pint`.

5.3 Related Work

We are not aware of existing work specifically on the unit canonicalization problem. The closest related works can be categorized as Semantic Web technologies

(Van Assem et al., 2010; Hignette et al., 2009; Samadian et al., 2014), regular expressions (Grecco, 2019; Wolfram|Alpha, 2019; Shbita et al., 2019), and Machine Learning (Lagi, 2016; Foppiano et al., 2019; Finkel et al., 2005), which are described below:

5.3.1 Semantic Web Technologies

A limited number of studies (Van Assem et al., 2010; Hignette et al., 2009) address how quantitative data columns can be annotated in terms of units, based on unit ontologies and a set of heuristics. For example, Van Assem et al. (2010) infer the unit of a data column by comparing the substrings of its header with unit symbols, and then employing simple heuristics such as symbols between brackets referring to units, e.g., “f (Hz)”. Similarly, Hignette et al. (2009) use a combination of two functions: (i) a cosine similarity function between the header and the units, such as Hertz, (ii) a function that assigns a non-zero score when a unit symbol is an element of the set of symbols for a unit. However, these methods do not apply to our task for several reasons. First, their annotations are based on ontological classes in domain-specific ontologies rather than the dimensions considered in this work. Second, the authors do not tackle the task of canonicalizing units of data entries. Third, the methods proposed in (Van Assem et al., 2010; Hignette et al., 2009) do not handle anomalous unit symbols and are not publicly available.

Chambers and Erwig (2010) consider the task of annotating data entries in spreadsheets rather than tabular data. The authors apply text processing techniques such as *tokenization* and *normalization* on headers for dimension and unit inference. They split the label in a header into separate words, remove word inflections and map word stems into known units and dimensions. Although their approach can be useful when information is explicitly given in a label, it does not use the information given implicitly, as in the label of “Credit Card Charges”.

Instead of focusing on annotating quantitative columns, Samadian et al. (2014) also consider the problem of canonicalizing units, but their focus is different from ours. The authors propose a Semantic Web Service-based approach by defining domain-specific ontological classes, e.g., “High-Systolic-Blood-Pressure-Measurement” class with `kilopascal` as its unit. Given a data column and its ontological class, their goal is to represent all entries with the same predefined unit of the corresponding ontological class. When a data entry has a unit different from the predefined unit, they send a query to Ontology of Units of Measure (Rijgersberg et al., 2013) to convert the data to the

predefined unit. The authors assume that symbols are always given in known forms that can be matched to the ontology. Hence, their proposed method does not handle out-of-vocabulary unit symbols that are common in messy datasets. Moreover, they consider only specific domains, preventing us from annotating data columns with the dimensions used in this study.

5.3.2 Regular Expressions

The majority of the previous research on units has focused on parsing and identifying units in text and unit conversion (Grecco, 2019; Wolfram|Alpha, 2019; Shbita et al., 2019), which are also not directly applicable to our task, as explained below.

Pint (Grecco, 2019) is a tool that can extract units from text based on regular expressions and knowledge about units. Moreover, it enables unit conversion based on the relationships among units. Similarly, Wolfram|Alpha (2019) presents an interface which can extract units from text and manipulate them, such as calculating the sum of two data values given in different units. Note that we have limited information about the methods behind Wolfram|Alpha since they are not explicitly described due to commercial concerns. Such tools can canonicalize different symbols that are commonly used to represent a unit when parsing text. However, they are designed neither to annotate quantitative data columns with dimensions nor to canonicalize the units of their entries. They do not even take the input as a data column, except for the professional version of Wolfram|Alpha which is not freely available.

For tabular data, Shbita et al. (2019) develop a rule-based system named CCUT by combining a grammar parser called Arpeggio and the unit ontology of QUDT. Their goal is to map unit symbols in tabular data to the ontology so that the data entries can be annotated with semantic information, which can be useful for table understanding. Unlike us, the authors do not use the contextual information in the entries of a data column, i.e., the unit symbols in the entries of a data column may be related to each other through the column dimension. Moreover, they do not consider the task of annotating quantitative data columns with dimensions.

5.3.3 Machine Learning

Quantulum (Lagi, 2016) combines regular expressions and knowledge graphs with a Machine Learning (ML) model in order to disambiguate unit symbols in unstructured text (e.g., whether “pound” in a sentence refers to currency or mass). Specif-

ically, Quantulum employs a linear Support Vector Machine (SVM) classifier on the character-level features (e.g., n-grams), which can result in a good performance on text data. However, this strategy may not be the best way to model tabular data due to two reasons: (i) it discards the context shared among the entries of a data column, (ii) the entries of a data column consist of only numeric values and unit symbols, unlike long sentences where additional information is available through the other words. Similarly, Foppiano et al. (2019) propose an ML framework named Grobid-Quantities (GQ), where Conditional Random Fields (CRFs) are used to parse and identify measurements in the scientific literature. GQ can be useful for processing text documents such as PDF files. However, identifying units in general tabular data can be more challenging since units are more likely to be reported with standard unit symbols in the scientific papers. In contrast, general tabular data may contain non-standard encodings of measurements.

Williams et al. (2020) tackle dimension and unit inference for spreadsheets rather than tabular data. Williams et al. (2020) combine logical constraint solving with ML, where the constraints are generated based on the format of the data values, spreadsheet formulas and tables. Note that tabular data such as CSV files do not support formulas, which prevents their proposed method from using the dependencies between cells. When no constraints are generated for a cell, the proposed method relies on regular expressions and simple ML techniques for dimension inference (e.g., the cosine similarities between word-embeddings representations of the header and the units), and then assign the most frequent unit for the inferred dimension as the column unit. Although this approach can be useful for dimension inference, it may be misleading for column unit inference as less frequent units can also be used to encode measurements in data columns. Moreover, its performance may be limited for unit inference as it does not handle anomalous unit symbols.

Lastly, dimension inference could benefit from Named Entity Recognition (NER) models. Existing NER models including Stanford NER (Finkel et al., 2005) are typically used to label entity mentions (e.g., San Diego) in text with tags (e.g., Location). However, the set of tags supported by Stanford NER includes only currency and time⁵. Moreover, it does not consider measurements encoded by numeric values and unit symbols as in tabular data. NER models can be trained by treating unit symbols as entity mentions and dimensions as their tags. However, training NER models for dimensions

⁵See <https://nlp.stanford.edu/software/CRF-NER.shtml> for a complete list of the supported tags.

can be challenging, given the limited availability of data.

5.4 Experiments

In this section, we first describe our experimental setup (Sec. 5.4.1) and then present quantitative results on column dimension inference, unit identification and unit canonicalization (Sec. 5.4.2). The goal of our experiments is to evaluate the robustness of the methods against real-world messy tabular datasets.

5.4.1 Experimental Setup

We describe the datasets, baseline methods and evaluation metrics used in our experiments below:

Datasets

We conduct experiments on 24 data columns obtained from 16 CSV data files, each of which contains at least one column where the measurements are encoded by units of measurement. The dimensions of the data columns were annotated by hand for these sets, resulting in 2 currency, 2 data storage, 6 mass, 3 volume and 11 length columns. We also annotated each data entry in terms of its numeric value and unit symbol. Note that numeric values are missing in 5 data columns, i.e., only unit symbols are observed in the entries. For example, the “Quantity Units” column of the Open Units dataset consists of three unique values. These are `ml`, `pint` and `cl`, which respectively denote the units of millilitres, pint and centilitres. Table 5.3 presents the number of entries, unique entries, units and unit symbols per data column. The number of unit symbols per data column varies between 2 and 11. Lastly, we have annotated the data columns for unit canonicalization evaluations by generating their clean versions, where we represent a data column with a single unit and scale its entries according to the annotated column unit when necessary.

Baselines

As we describe in Sec. 5.3, the methods proposed in (Van Assem et al., 2010; Hignette et al., 2009; Samadian et al., 2014; Shbita et al., 2019; Foppiano et al., 2019; Grecco, 2019; Lagi, 2016; Finkel et al., 2005) do not address the unit canonicalization task. However, we can construct baselines to be used in our experiments by adapting some

dataset	column	# non-missing entries	# unique entries	# units	# unit symbols
Arabica	Bag W...	1,283	51	2	2
Arabica	Altitude	132	81	2	11
HES	Freezer...	50	47	2	11
HES	Refrig...	38	37	2	9
Huffman	DIST...	71	68	2	2
Maize	PACK...	30	10	2	2
MBA	CURR...	43	19	3	6
Open U...	Quanti...	1,082	3	3	3
PHM	Height	30,312	1,313	3	5
PHM	Weight	179	135	2	3
PHM	Width	36,330	1,282	3	5
PHM	Depth	19,400	1,073	3	4
PHM	Diameter	2,106	377	2	2
Robusta	Bag W...	28	4	2	2
Robusta	Altitude	8	3	1	2
query_2	unitH...	22	3	3	3
query_2	unitW...	22	3	3	3
query_4	unitH...	22	3	3	3
Zomato	currency	6,386	9	9	9
143...62	FORMAT	123	19	2	3
143...23	Size	3,855	1,667	2	2
228...96	Size	377	256	2	2
3b5...ff	amount	5	5	2	2

Table 5.3: Size of the datasets used and the number of units and unit symbols in each data column.

of these methods, namely CCUT (Shbita et al., 2019), Grobid-Quantities (GQ) (Foppiano et al., 2019), Pint (Grecco, 2019), Stanford NER (S-NER, Finkel et al. 2005) and Quantum (Lagi, 2016). All these methods take as input a sentence (e.g., "... 2 litres of water.") and annotate the words that refer to quantities (e.g., 2 litres) with their dimensions (e.g., volume), except S-NER which needs to be trained again for the dimension prediction. Here, we use the data values of the entries as inputs to these methods.

We construct baselines for dimension inference as follows. We first identify the dimensions of the entries of a data column using a competitor method and then predict the column dimension through majority voting, i.e., assigning the most common

dimension as the column dimension. The baseline method based on the S-NER is obtained by training the pre-trained model with pairs of unit symbols and their dimensions (e.g., `metres` - `length`). The resulting model then generates a tag for each data entry of a data column and can be used to assign a dimension to that column through majority voting.

On the unit identification task, we evaluate whether the unit of a data entry can be correctly identified, e.g., `1 cm` as `1` and `centimetre`. For this, we compare our method with CCUT, GQ, Pint and Quantulum, which are developed for identifying units in textual documents.

Lastly, on the unit canonicalization task, we report the performance of our method for canonicalizing different unit symbols in data columns. We compare our method with a baseline constructed based on Pint, where the column unit is assumed to be known.

Evaluation Metrics

We use different sets of metrics for dimension inference, unit identification and unit canonicalization. For dimension inference, we use the overall accuracy and the Jaccard index described in Chapter 3. See Section 3.3.1 for a detailed description of these metrics and how they are used for type inference. Additionally, we evaluate the runtime of each method per data column. To measure the performance on the unit identification task, we report the accuracies of the methods per dataset and apply paired t-tests to determine whether the predictions of the competitor methods are significantly different from the predictions of our method. Note that the accuracy on the unit identification task measures the ratio of number of entries in a dataset for which the unit is correctly predicted over the number of entries in a dataset. Lastly, we report the accuracies of the methods per data column on unit canonicalization, which indicates the ratio of number of correctly canonicalized entries in a data column over the number of its entries.

5.4.2 Results

We present quantitative results on three tasks: (i) column dimension inference, (ii) unit identification, and (iii) unit canonicalization. As we describe in Sec. 5.4.1, we measure the performance of the methods on: (i) predicting dimensions of data columns in column dimension inference, (ii) identifying the magnitude and unit symbol of a data entry, e.g., `1 cm` as `1` and `centimetre` in unit identification, and (iii) carrying out unit

canonicalization.

Column Dimension Inference

Table 5.4 presents the performance of the methods on the column dimension inference task. The overall accuracies show that ptype-semantics performs better than the competitor methods. We observe a similar trend with the performance per dimension, quantified through the Jaccard index. These improvements are due to our model’s extensive knowledge about units and its structure that takes into account the context shared among data rows. Note that Jaccard index becomes zero when a method incorrectly labels all the data columns of a particular dimension. As per Table 5.4, Jaccard index for the currency dimension is obtained as zero by CCUT, which is primarily because it cannot properly parse the corresponding data values. For example, CCUT cannot handle the parenthesis signs in the data values of Dollar (\$) and Pounds (£) which occur in the entries of the “Currency” column of the Zomato dataset.

Method \ Dimension	CCUT	GQ	Pint	S-NER	Quantulum	ptype-semantics
Currency	0.00	0.00	0.00	0.67	0.50	1.00
Data storage	0.00	0.00	0.00	0.00	0.50	1.00
Length	0.27	0.45	0.67	0.62	0.58	0.91
Mass	0.00	0.67	0.57	0.67	1.00	1.00
Volume	0.00	0.00	0.00	0.33	0.67	1.00
Overall Accuracy	0.12	0.38	0.50	0.71	0.71	0.96

Table 5.4: Performance of the methods for dimension inference using the Jaccard index and overall accuracy, for the dimensions Currency, Data storage, Length, Mass and Volume. We highlight the best score in each row by making the highest score bold.

ptype-semantics correctly predicts the dimensions of all data columns, except one data column for which all the competitor methods also fail. These failures result from the inability to parse the data values such as 5'10" where ' and " denote respectively feet and inches. We could improve our regular expression to parse such formats, which we have not done in order not to optimise on test datasets. On the other hand, the leading competitor methods are S-NER and Quantulum, which achieve the same overall accuracy. They both fail to identify the column dimensions of seven data columns. To compare our method with these baselines, we present their normalised confusion ma-

trices in Fig. 5.2, normalised so that a column sums to 1 except the right-most column of each matrix.

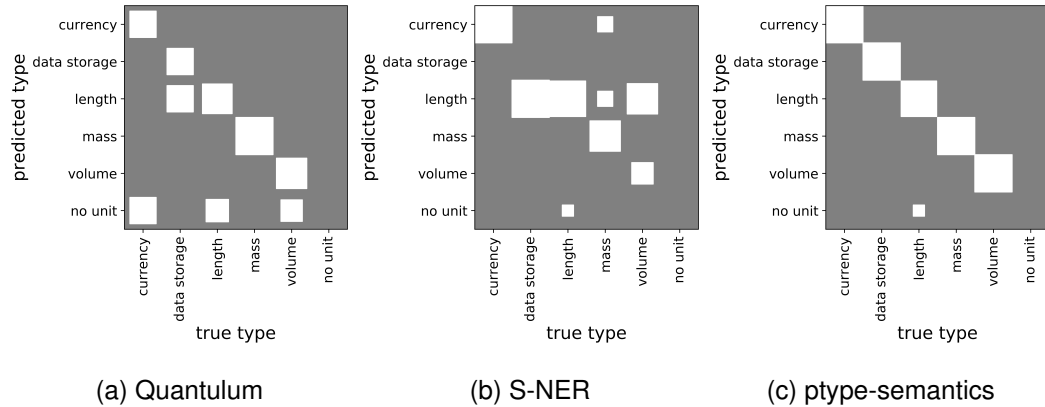


Figure 5.2: Normalized confusion matrices for (a) Quantum, (b) S-NER and (c) ptype-semantics plotted as Hinton diagrams, where the area of a square is proportional to the magnitude of the entry.

Fig. 5.2(b) shows that S-NER tends to infer the column dimension as length. These failures can be explained by the differences in the number of characters of unit symbols. S-NER performs better on longer unit symbols, which may not be surprising as it uses n-grams for feature extraction. For example, the confusions between volume and length occur on the HES dataset, where the dimensions of units symbols for cubic foot (e.g., cuft, cu.ft) are correctly predicted as volume. Nevertheless, the dimensions of unit symbols for litre (e.g., l, L) are predicted as length instead of volume, which result in incorrect predictions since they are more frequent in the data. Character-level features (e.g., n-grams) could be useful to handle variations in the data such as misspellings; however, they may lead to limited performance on short unit symbols. This result indicates the advantage of incorporating knowledge about unit symbols directly into the model, as in ptype-semantics.

As we discuss above, Quantum cannot parse measurements such as 5'10" and fails to identify the dimension of the corresponding data column. The confusion between length and data storage occurs in a data column where kilobyte and megabyte are respectively encoded by K and M. Here, Quantum mislabels M as metre and does not generate a prediction for K. A detailed inspection of the remaining five columns shows that numeric values are missing in the entries, which cannot be handled by Quantum.

To determine whether the column dimension predictions of ptype-semantics and the leading competitor methods (namely S-NER and Quantum) are significantly dif-

ferent, we apply a variation of the McNemar’s test as the number of samples is low (see e.g., Edwards (1948)). This test assumes that the two methods should have the same error rate under the null hypothesis. We compute the exact p-value $2 \sum_{i=n_{10}}^{n_{01}} \binom{n}{i} 0.5^i (1 - 0.5)^{n-i}$ where $n = n_{01} + n_{10}$ with n_{01} and n_{10} which respectively denote the number of columns misclassified by only a competitor method, and by only ptype-semantics. The test to compare ptype-semantics and S-NER results in a p-value of 0.03 since n_{01} and n_{10} are respectively equal to 6 and 0. These results reject the null hypothesis that the means are equal and confirm that they are significantly different at the 0.05 level. We obtain the same result from the test between ptype-semantics and Quantumum, as n_{01} and n_{10} are the same as S-NER, which confirms that they are significantly different at the 0.05 level.

Lastly, we have evaluated the runtime of each method per data column. Figure 5.3 shows that S-NER is the slowest method, whereas ptype-semantics is the fastest on average. The leading competitor method is Pint; however, the variation in its runtime is higher than ptype-semantics. Note that we have not explicitly optimised our method for speed, which may improve further its scalability.

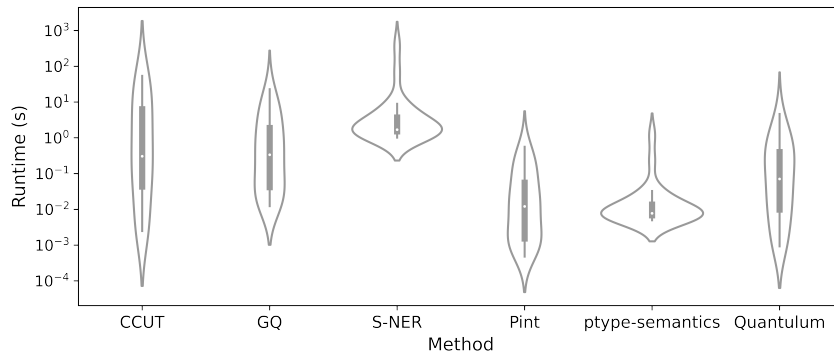


Figure 5.3: Runtime violin plots denote the time in seconds taken to infer dimensions per column. The dot, box, and whiskers respectively denote the median, interquartile range, and 95% confidence interval.

Unit Identification

Table 5.5 presents the accuracy of each method on each dataset (aggregated over columns) and its overall accuracy calculated by averaging over all datasets. We observe a similar trend as before in that ptype-semantics performs consistently better than the competitor methods, whereas the competitor methods are competitive with

our method on some of the datasets. Note that accuracy becomes zero when a method fails to identify the unit of any data entry of a dataset. As per the table, Pint leads to the accuracy of zero for the MBA dataset because it cannot recognize the corresponding unit symbols. For example, Pint cannot recognize the unit symbols of LB and OZ in the entries of the MBA dataset, which respectively denote the pound and ounce units.

Method \ Dataset	CCUT	GQ	Pint	Quantulum	p _{type} -semantics
Arabica	0.77	0.17	0.66	1.00	0.70
HES	0.18	0.00	0.27	0.64	0.98
Huffman	0.06	0.53	1.00	1.00	1.00
Maize	1.00	0.30	1.00	1.00	1.00
MBA	0.00	0.00	0.00	0.84	0.95
Open Units	0.00	0.00	1.00	0.00	1.00
PHM	0.99	0.95	0.99	1.00	1.00
query_2	0.00	0.00	1.00	0.00	1.00
query_4	0.00	0.00	1.00	0.00	1.00
Robusta	0.71	0.43	1.00	1.00	1.00
Zomato	0.00	0.00	0.00	0.00	0.60
143...23	0.00	0.69	0.00	0.00	0.97
143...62	0.58	0.00	0.00	0.95	0.95
228...96	1.00	0.00	1.00	1.00	1.00
3b5...ff	0.60	0.00	0.00	1.00	1.00
Overall Accuracy	0.39	0.20	0.59	0.62	0.94

Table 5.5: Accuracy of the methods on unit identification. We highlight the best score in each row by making the highest score bold.

p_{type}-semantics outperforms the competitor methods by a large margin on 4 datasets (143...23, HES, MBA and Zomato) out of the 15. Note that we exclude the Taser dataset from the evaluations since none of the methods could parse its values such as 5'10" where ' and " denote respectively feet and inches. On the remaining 11 datasets, there is at least one competitor method competitive with ours.

The performance gap between p_{type}-semantics and the competitor methods reflects the importance of mapping anomalous unit symbols to known symbols through string-similarity. For example, on the HES dataset, the competitor methods could accurately identify only a few unit symbols, whereas our method could successfully identify al-

most all of the unit symbols. Out of 14 unique unit symbols, CCUT identified `L` and `cuft`, and Pint identified `L`, `l` and `litres`. In addition to these three unit symbols, Quantum identified `Litres`. Surprisingly, GQ, which is one of the state-of-the-art methods in identifying units in text documents, could not identify any of these unit symbols. `p`-type-semantics, on the other hand, could identify 12 unit symbols correctly, with only two unidentified unit symbols (`Cu` and `cf`). We observe that Quantum performs better than `p`-type-semantics on the Arabica dataset. This result is mainly due to the Altitude column where metre is encoded by `M`, which is a known symbol for mile. Consequently, our method predicts the units of such entries as mile rather than metre. We could avoid this confusion by making row units dependent, so that the presence of unit symbols (e.g., `m`, metres) in the other data entries that encode the same unit (e.g., metre) is treated as an indicator of `M` being a symbol for metre rather than mile. Here, we do not adapt our model accordingly so that it is not optimised on test datasets.

To determine whether the performances of `p`-type-semantics and Pint are significantly different, we have applied a paired t-test on the differences of the accuracies, i.e., $\text{Accuracy}(\text{p-type-semantics}) - \text{Accuracy}(\text{competitor method})$. Table 5.6 presents these results, which reject the null hypothesis that the means are equal and confirm that they are significantly different at the 0.05 level.

Method	CCUT	GQ	Pint	Quantulum
t-statistic	4.85	8.61	2.74	2.55
p-value	0.0002	0.000001	0.01	0.02

Table 5.6: The t-statistics and p-values obtained by applying a paired t-test on the differences of the accuracies, i.e., $\text{Accuracy}(\text{p-type-semantics}) - \text{Accuracy}(\text{competitor method})$.

Lastly, Table 5.5 indicates that Pint and Quantum perform best among the competitor methods. We have compared their performances through a paired t-test, i.e., $\text{Accuracy}(\text{Quantulum}) - \text{Accuracy}(\text{Pint})$, which suggests that they are not significantly different at the 0.05 level (the t-statistics of 0.18 and the p-value of 0.86). This result suggests that neither method is comprehensive enough to handle various unit symbols observed in different datasets.

Unit Canonicalization

Table 5.7 presents the accuracies of ptype-semantics and Pint on each data column as well as their overall accuracies calculated by averaging over the data columns. The results indicate that ptype-semantics either outperforms or is equally good as Pint on all data columns except one. Note that accuracy becomes zero when a method fails to canonicalize any entry of a dataset. For example, ptype-semantics leads to the accuracy of zero for the “Altitude” column of the Arabica dataset as it cannot recognize the corresponding unit symbols, as discussed in Sec. 5.4.2).

We exclude seven columns from the evaluations. The data entries of two columns (from the 3b5...ft and Zomato datasets) contain unit symbols used to denote currency, for which conversion rates change dynamically over time. In such cases, our model can inform the analyst of the dimensions so that the rate of conversion between currencies can be given manually. Moreover, the data entries of the four columns obtained from three datasets (Open Units, query_2 and query_4) do not contain any numeric values, preventing us from converting units. Note that the numeric values are available in other data columns. Our model can notify the analysts of the missing numeric values so that the corresponding values can be provided to convert the units. Similarly to the unit identification evaluations, we exclude the Taser dataset as the methods could not parse the entries of its column.

ptype-semantics outperforms the competitor method Pint by a large margin on 6 data columns out of 17. Both methods achieve perfect results on the 9 columns whereas pint excels at the remaining one. The performance of a method on unit canonicalization is limited by its ability to identify units correctly. Consider the “Freezer_volume” column of the HES dataset where the data entries contain 11 different unit symbols. As we discuss in Sec. 5.4.2, ptype-semantics correctly identifies a higher number of units than Pint on the HES dataset. As a result, more units are correctly canonicalized by ptype-semantics, i.e., ptype-semantics canonicalizes all unit symbols except `Cu` and `cf`, whereas Pint could only canonicalize `L`, `l` and `litres`.

The data column where Pint performs better than ptype-semantics is the Altitude column of the Arabica dataset, where ptype-semantics incorrectly identifies the unit of the symbol `M`, as we discuss in Sec. 5.4.2. This failure leads to incorrect unit conversions. See Sec. 5.4.2 for a discussion of how the performance of our model can be improved for such cases.

To determine whether the performances are significantly different, we have applied

Dataset	Column	Pint	ptype-semantics
Arabica	Bag Weight	1.00	1.00
	Altitude	0.44	0.00
HES	Freezer_volume	0.26	0.96
	Refrigerator_volume	0.30	1.00
Huffman	Distance	1.00	1.00
Maize	Pack Size	1.00	1.00
MBA	Product Size	0.00	0.95
PHM	Height	1.00	1.00
	Weight	0.61	1.00
	Width	1.00	1.00
	Depth	1.00	1.00
	Diameter	0.99	1.00
Robusta	Bag Weight	1.00	1.00
	Altitude	1.00	1.00
143...23	Size	0.00	0.97
143...62	Format	0.00	0.95
228...96	Size	1.00	1.00
Overall Accuracy		0.68	0.93

Table 5.7: Accuracies of the methods on handling unit canonicalization problems per data column.

a paired t-test on the differences of the accuracies, i.e., $\text{Accuracy}(\text{ptype-semantics}) - \text{Accuracy}(\text{Pint})$. We have calculated the t-statistic of 2.13 and the p-value of 0.04. These results reject the null hypothesis that the means are equal and confirm that they are significantly different at the 0.05 level.

5.5 Discussion

Syntactic type inference methods including ptype do not make use of the semantic information about a given dataset. However, such information may be useful for accurate and enhanced type inference. In this chapter, we demonstrate how ptype can be enriched semantically by using knowledge graphs, and tackle non-standard encodings of measurements in quantitative data columns where otherwise time-consuming manual data cleaning efforts would be needed. The resulting model called ptype-semantics can automatically identify the units of the entries in a data column, predict the column dimension and canonicalize its entries. Our experiments on real-world data demonstrate that the proposed model is consistently superior to competing techniques. Moreover,

we make these tabular datasets and our annotations regarding units of measurement available with the aim of accelerating research in this area.

We can extend ptype-semantics further. For example, we can assume that the numeric values and units in the entries of a data column are related. This assumption may hold in certain scenarios, e.g., height measurements of people where one would expect heights of adults in the range 150-250 cm, and thus 1.50 to 2.50 metres. In such cases, incorporating the numeric values $\{v_i\}_{i=1}^N$ into our model may enable us to address certain problems such as *unit imputation*, which is the task of identifying the unit of a data entry for which no unit symbol is present, and *unit repair*, which is the task of identifying the unit of a data entry where the measurement is encoded by an anomalous unit symbol.

Another interesting research direction is to investigate how meta-data such as headers can be taken into account as an additional source of information for dimension and unit inference. We believe that it would sometimes be possible to identify the dimension and unit of a data column based on its header, similarly to how data type was inferred from meta-data in Chapter 4.

Chapter 6

Conclusions and Future Work

In this chapter, we summarize the contributions of this thesis (Sec. 6.1) and then discuss possible future work (Sec. 6.2)

6.1 Summary of the Contributions

Type inference is challenging primarily because raw datasets often suffer from the presence of missing and anomalous data entries. This thesis has focused on developing probabilistic models that allow us to identify types accurately, even for such messy datasets. Moreover, our models provide additional capabilities to semi-automate certain data preprocessing tasks, including handling categorical variables and non-standard encodings of measurements, which would need to be manually resolved by data scientists otherwise.

Firstly, in Chapter 3, we proposed `ptype` - a novel probabilistic model for syntactic type inference, which is typically carried out by using Finite-State Machines (FSMs), such as regular expressions, that either accept or reject a given data value. In contrast, `ptype` uses Probabilistic Finite-State Machines (PFSMs) that assign probabilities to different values and therefore offer the advantage of generating weighted predictions when a column of messy data is consistent with more than one type assignment. Moreover, `ptype` allows us to identify any values which (conditional on the inferred column type) are deemed missing or anomalous.

We compared `ptype` with applicable existing approaches (Trifacta, 2018; Petricek et al., 2016; Lindenberg, 2017; Stochastic Solutions, 2018; Döhmen et al., 2017; Wickham et al., 2017) using a variety of datasets. Our evaluations demonstrated the advantages of using probabilistic variants of Finite-State Machines and the importance of

explicitly modeling the missing and anomalous data for accurate type inference. Note that we also provide a software package for `ptype` with support for a number of interactions with the users.

Syntactic type inference methods such as `ptype` can classify Boolean variables as categorical; however, they treat non-Boolean categorical variables as either integers or strings rather than categoricals. Therefore, the user needs to transform these data columns into categorical and specify the corresponding categorical values manually, which can be tedious and time-consuming. In Chapter 4, we proposed an alternative approach that can eliminate the need for this manual work. To better detect the categorical type (including non-Boolean variables), we extended `ptype` by combining its output with additional features that can indicate whether a column denotes a categorical variable or not, and employed a separate binary classifier to identify the categorical variables. In addition, we identified the categorical values of a categorical variable by adapting `ptype`. We showed that our probabilistic approach can be more flexible than hard-choices made by existing applicable methods (Bot, OpenML and Weka) to infer the type of a data column. Moreover, we investigated the use of meta-data for inferring data types and categorical values. We observe that meta-data has the potential to slightly improve the performance for type and categorical value inference when combined with tabular data-based approaches. However, it is still possible to obtain a good overall performance based on tabular data itself without extracting text sequences from meta-data.

`ptype` relies on the syntax of the data values for type inference, whereas `ptype-cat`, in addition to the syntax, takes into account additional features to better detect the categorical type. However, they do not make use of the semantic information about a given dataset, which can be essential for accurate and enhanced type inference. In Chapter 5, we presented `ptype-semantics` which is an extension of `ptype` enriched semantically by using knowledge graphs about units of measurement. The proposed model allows us to extract semantic information about a given data column (such as its dimension and unit), to canonicalize its entries, and it provides enhanced type inference capabilities for syntactic type inference methods such as `ptype`.

`ptype-semantics` showed that Knowledge Graphs can be useful for enhanced type inference and accelerating certain data preprocessing efforts such as the canonicalization of units. Our experiments indicated that `ptype-semantics` performs better than other applicable solutions (Shbita et al., 2019; Foppiano et al., 2019; Grecco, 2019; Finkel et al., 2005; Lagi, 2016).

6.2 Future Work

We envision several ways to extend our work. Below we describe four open research questions that would be useful to investigate further.

6.2.1 Identifying the Ordinal and Nominal Types

There are two types of categorical variables: (i) ordinal variables that have ordered categorical values (e.g., patient status with the categorical values of good, fair, serious and critical) and (ii) nominal variables for which there is no intrinsic ordering for the categorical values (e.g., genre of music with the categorical values of classical, folk, jazz and rock). See Agresti (2003) for a detailed description. Although `pctype-cat` can automatically detect categorical variables, it does not distinguish between ordinal and nominal variables, which are included within the categorical type. Therefore, the analyst would need to manually make such distinctions which may be useful for the later stages of the data analytics, such as data modeling.

One future research direction is investigating how `pctype-cat` can be improved to determine whether a categorical variable represents an ordinal or a nominal variable. This can potentially be accomplished by feeding `pctype-cat` with new features extracted from categorical values themselves. For example, Hernández-Lobato et al. (2014) model *numeric* categorical data with different observation models, and discriminate between the ordinal and nominal types by comparing the model evidence and the predictive test log-likelihood of ordinal regression models and multi-class classifiers. One could run such models on data and use their outputs as features for `pctype-cat`.

Alternatively, one could encode each of the categorical values encoded as *strings* with word-embeddings, and then combine them with a set transformer network (see e.g., Lee et al., 2019) which is invariant (as required) to permutations of the input order. This could then be used as input to an ordinal/nominal classifier.

6.2.2 Using Header Information to Aid Inference

In Chapter 4, we investigated how meta-data can be used to infer data types and categorical values. In particular, we extracted a text sequence for each data column and employed sequence classification and named entity recognition models. An additional source of information is the header, which is also known as the label. For example,

the header of the National Football League scouting dataset¹ has the labels of “Height (in)” and “Weight (lbs)” where `in` and `lbs` respectively denote the inch and pound units. However, the unit symbols can be missing as in “Credit Card Charges”. In such cases, it may still be possible to identify the dimension (which is currency in this case) based on the semantics in the header.

Chambers and Erwig (2010) and Williams et al. (2020) take advantage of labels in spreadsheets for dimension and unit inference. Chambers and Erwig (2010) split a label (e.g., “Total Gallons”) into separate words (e.g., “Total” and “Gallons”), remove word inflections (e.g., converting “Gallons” into “Gallon”) and map these word stems into known units and dimensions (e.g., Gallon and volume). Their approach can identify units and dimensions when information is explicitly given in a label. However, it does not use the information given implicitly, as in the label of “Credit Card Charges”. Similarly, Williams et al. (2020) address the detection of explicitly given units in labels by using pre-defined templates (e.g., checking the presence of unit symbols inside parenthesis as in “Length (m)”). In addition, they rely on word-embeddings to carry out dimension and unit inference when information is given implicitly (e.g., inferring the dimension of the label “Credit Card Charges” as currency, and then assigning the most frequent currency unit, which is dollar, as its unit). The idea is to calculate the cosine similarity between the word-embedding representation of known units and labels, and then assign the closest dimension as the column dimension. It may be possible to extend our model to use the information in a similar manner. Another interesting research direction is to investigate the use of labels in different columns for dimension inference. This may allow us to incorporate the context of the whole dataset for inferring the dimension of a particular column (e.g., the height and length dimensions may co-occur more often than the height and currency dimensions do).

6.2.3 Using Numeric Values for Unit Inference

ptype-semantics relies on the presence of unit symbols in the entries of a data column to infer the semantics of a data column. However, numeric values in the entries of a data column can also be informative about the semantics when the numeric values and units are related, e.g., height measurements of people where one would expect heights of adults in the range 150-250 cm, and thus 1.50 to 2.50 metres. In such cases, incorporating the numeric values into our model may enable us to address certain problems

¹The data is publicly available at <https://www.kaggle.com/dtrade84/2019-nfl-scouting-combine>

such as *unit imputation*, which is the task of identifying the unit of a data entry for which no unit symbol is present, and *unit repair*, which is the task of identifying the unit of a data entry where the measurement is encoded by an anomalous unit symbol.

We now present a simple application to demonstrate how our model can be used to impute missing units. Given a set of numeric values $\mathbf{v} = \{v_i\}_{i=1}^N$ and their units $\mathbf{u} = \{u_i\}_{i=1}^N$, the task is to predict the units of the data entries for which no unit symbols are present, i.e., $z_i = m$. We cast this problem as the problem of calculating the posterior distribution of the missing units given its numeric value, namely $p(u_i | v_i, z_i = m)$. Consider the 1438042987662 dataset, which is a list of bakery ingredients. The measurements in the “FORMAT” column are encoded by the unit symbols of `KG` and `LB`, which respectively denote kilograms and pounds. Note that `KG` is used in 86 data entries, whereas `LB` is used in 36 data entries. Although the numeric values overlap (i.e., they vary between 1 and 38 for kilograms, and 18 and 50 for pounds), their distributions, which are shown in Fig 6.1 (left), confirm our assumption that numeric values and units are related, and can therefore be helpful for unit inference when the unit information is missing.

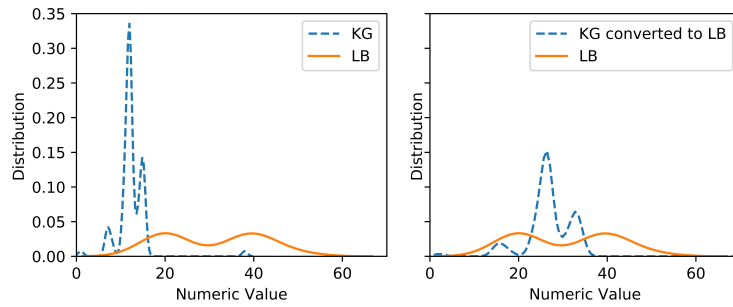


Figure 6.1: The probability density function of the numeric values in the original raw data according to their units (left) and the probability density function of the numeric values in the scaled data after kilogram-pounds conversion (right). The density plots are obtained using a kernel density estimator².

6.2.4 Enhancing User Interactions

We have made our implementation of `pptype` publically available as a Python package in order to enhance the impact of our work. This tool can be used to infer a schema for a given dataset and to transform it into its “clean” version based on this schema.

²We used Gaussian kernels with the bandwidths of 0.41 and 0.49 respectively for `KG` and `LB`.

These operations can reduce the time and effort spent to manually transform the data; however, it requires user interaction when the inferred schema contains incorrect type predictions. Therefore, we considered a set of scenarios where the schema is inferred incorrectly, and demonstrated how the user can interact with the tool to handle the errors (see Section 3.3.3 for a detailed discussion). For example, the user can choose the correct type when `p`type labels a column with an incorrect type. Similarly, `p`type may incorrectly treat some values as missing. The user can handle such cases by updating the set of values that `p`type is expected to treat as missing and then re-running the inference.

We are investigating alternative user interaction styles for `p`type. In this regard, one choice is to design a rejection scheme where the user would tell the tool that a particular type annotation is incorrect, allowing the next most likely type to be populated as the corresponding type prediction. This type of feedback mechanism can also be used for missing and anomalous values. User feedback would also be needed for `p`type-cat. In fact, there may be new challenges resulting from detection of the general categorical type. For example, the user may want to reject annotated categorical values or manually specify the correct categorical values. We plan to present the tool based on `p`type-cat as an extension to the `p`type package. Finally, it may be possible to improve `p`type-semantics through user feedback (e.g., the user can manually identify the unit of an unknown unit symbol). It would be useful to feed such information back into our knowledge graph about units of measurement.

Appendix A

Appendix for ptype

In this Appendix, we discuss the implementation of the Probabilistic Finite State Machines (PFSMs) (Appendix A.1), describe the data sets used (Appendix A.2) and present the derivations for training in our model (Appendix A.3). Moreover, we discuss the behavior of PADS with an example (Appendix A.4) and report scalability of the methods (Appendix A.5).

A.1 PFSMs for Data Types

In this work, we use five regular data types including integers, strings, floats, Booleans, and dates; and two noisy data types, namely missing and anomaly.

A.1.1 Integers

Please see Sec. 3.2.1 for a detailed discussion of the PFSM used to represent integers.

A.1.2 Floats

A floating-point number often consists of digits and a full stop character, which is followed by another set of digits. However, they can also be written without any fractional component, i.e. as integer numbers. We also support the representations of floating-point numbers with `e` or `E`. Lastly, we support the use of comma for the thousands separator in floating-point numbers, such as `1,233.15`, `1,389,233.15`, etc.

A.1.3 Strings

The string PFSM is constructed with one initial state and one final state. Through each transition, either a digit, an alpha character, or a punctuation character is emitted. The punctuation characters considered here are `.`, `,`, `-`, `_`, `%`, `:`, and `;`, which are commonly found in real-world data sets to represent columns with the string type.

A.1.4 Booleans

Our machine supports the following values by assigning them non-zero probabilities: Yes, No, True, False, 1, 0, -1 and their variants yes, Y, y, no, true, false.

A.1.5 Dates

We categorize date formats into two groups, which are detailed below:

ISO-8601

We support values in *YYYY-MM-DDTHH:MM::SS*, where T is the time designator to indicate the start of the representation of the time of day component. We also support other ISO-8601 formats such as *YYYYMMDD*, *YYYY-MM-DD*, *HH:MM*, and *HH:MM:SS*.

Nonstandard Formats

We treat years in *YYYY* format as date type. To distinguish years and integers, we restrict this to the range of [1000-2999]. On the other hand, we do not explicitly constrain the month (*MM*) and day columns (*DD*) to valid ranges, and but treat them as integers. We support ranges of years with the formats of *YYYY-YYYY*, *YYYY YYYY*, *YYYY - YYYY*, *YYYY -YYYY*, and *YYYY- YYYY*. Lastly, We support dates written as *MM-DD-YYYY HH:MM:SS AM/PM*, and months, e.g., January, February, etc.

A.1.6 Missing

The machine for missing data assigns non-zero probabilities to the elements of this set, including Null, NA and their variants such as NULL, null, “NA ”, NA, “N A”, N/A, “N/A”, “N /A”, “N /A”, N/A, #NA, #N/A, na, “ na”, “na ”, “n a”, n/a, N/O, NAN, NaN, nan, -NaN,

and `-nan`; special characters such as `-`, `!`, `?`, `*`, and `.`; integers such as `0`, `-1`, `-9`, `-99`, `-999`, `-9999`, and `-99999`; and characters denoting empty cells such as `""` and `" "`.

A.1.7 Anomaly

We use all of the Unicode characters in this machine’s alphabet, including the accented characters. Note that the number of elements in this set is 1,114,112.

A.2 Data Sets

We share the available data sets and the corresponding annotations at <https://github.com/google-research/google-research>. Here, we briefly describe these data sets, and provide a list in Table A.1 which denotes their sources, and whether they are used in the training or testing phase.

- **Accident 2016:** information on accidents casualties across Calderdale, including location, number of people and vehicles involved, road surface, weather conditions and severity of any casualties.
- **Accidents 2015:** a file from Road Safety data about the circumstances of personal injury road accidents in GB from 1979.
- **Adult:** a data set extracted from the U.S. Census Bureau database to predict whether income exceeds \$50K/yr.
- **Auto:** a data set consisting of various characteristics of a car, its assigned insurance risk rating, and its normalized losses in use.
- **Broadband:** annual survey of consumer broadband speeds in the UK.
- **Billboard:** a data set on weekly Hot 100 singles, where each row represents a song and the corresponding position on that week’s chart.
- **Boston Housing:** a data which contains census tracts of Boston from the 1970 census.
- **BRFSS:** a subset of the 2009 survey from BRFSS, an ongoing data collection program designed to measure behavioral risk factors for the adult population.
- **Canberra Observations:** weather and climate data of Canberra (Australia) in 2013.

- Casualties 2015: a file from Road Safety data about the consequential casualties.
- Census Income KDD: a data set that contains weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau.
- CleanEHR (Critical Care Health Informatics Collaborative): anonymised medical records ¹.
- Cylinder Bands: a data set used in decision tree induction for mitigating process delays known as “cylinder bands” in rotogravure printing.
- data.gov: 9 CSV files obtained from data.gov, presenting information such as the age-adjusted death rates in the U.S., Average Daily Traffic counts, Web traffic statistics, the current mobile licensed food vendors statistics in the City of Hartford, a history of all exhibitions held at San Francisco International Airport by SFO Museum, etc.
- EDF Stocks: EDF stocks prices from 23/01/2017 to 10/02/2017.
- El Niño: a data set containing oceanographic and surface meteorological readings taken from a series of buoys positioned throughout the equatorial Pacific.
- FACA Member List 2015: data on Federal Advisory Committee Act (FACA) Committee Member Lists.
- French Fries: a data set collected from a sensory experiment conducted at Iowa State University in 2004 to investigate the effect of using three different fryer oils on the taste of the fries.
- Fuel: Fuel Economy Guide data bases for 1985-1993 model.
- Geoplaces2: information about restaurants (from UCI ML Restaurant & consumer data).
- HES (Household Electricity Survey): time series measurements of the electricity use of domestic appliances (to gain access to the data, please register at <https://tinyurl.com/ybbqu3n3>).

¹a subset is available at https://github.com/ropensci/cleanEHR/blob/master/data/sample_ccd.RData

- **Housing Price:** a data set containing 79 explanatory variables that describe (almost) every aspect of residential homes in Ames, Iowa.
- **Inspection Outcomes:** local authority children's homes in England - inspection and outcomes as at 30 September 2016.
- **Intel Lab:** a data set collected from sensors deployed in the Intel Berkeley Research lab, measuring timestamped topology information, along with humidity, temperature, light and voltage.
- **mass.gov:** 4 CSV files obtained from mass.gov, which is the official website of the Commonwealth of Massachusetts.
- **MINY Vendors:** information on "made in New York" Vendors.
- **National Characteristics:** information on the overall, authorised, unauthorised and persistent absence rates by pupil characteristics.
- **One Plus Sessions:** information on the number of enrollments with one or more session of absence, including by reason for absence.
- **Pedestrian:** a count data set collected in 2016, that denotes the number of pedestrians passing within an hour.
- **PHM Collection:** information on the collection of Powerhouse Museum Sydney, including textual descriptions, physical, temporal, and spatial data as well as, where possible, thumbnail images.
- **Processed Cleveland:** a data set concerning heart disease diagnosis, collected at Cleveland Clinic Foundation (from the UCI ML Heart Disease Data Set).
- **Sandy Related:** Hurricane Sandy-related NYC 311 calls.
- **Reported Taser 2015:** a hand-compiled raw data set based on forms filled out by officers after a stun gun was used in an incident, provided by CCSU's Institute for Municipal and Regional Policy.
- **Rodents:** the information collected on rodents during a survey.
- **Survey:** a data set from a 2014 survey that measures attitudes towards mental health and frequency of mental health disorders in the tech workplace.

- TAO: a real-time data collected by the Tropical Atmosphere Ocean (TAO) project from moored ocean buoys for improved detection, understanding and prediction of El Niño and La Niña.
- Tb: a tuberculosis dataset collected by the World Health Organisation which records the counts of confirmed tuberculosis cases by “country”, “year”, and demographic group.
- Tundra Traits: measurements of the physical characteristics of shrubs in the arctic tundra.
- T2Dv2 Gold Standard: a set of data Web tables to evaluate matching systems on the task of matching Web tables to the DBpedia knowledge base.
- User Profile: information about consumers (from UCI ML Restaurant & consumer data).
- Vehicles 2015: a file from Road Safety data about the types of vehicles involved in the accidents.
- 83492acc-1aa2-4e80-ad05-28741e06e530: a hypoparsr data set which contains information on expenses.

Note that the data sets from mass.gov and data.gov are obtained from Abdulhakim A. Qahtan and also used in Qahtan et al. (2018).

name	source	training/test	# columns	# rows
Accidents 2015	data.gov.uk	test	32	140,056
Accident 2016	data.gov.uk	test	18	555
Adult	UCI ML	training	15	32,561
Auto	UCI ML	test	26	205
Broadband	data.gov.uk	training	55	2,732
Billboard	github.com	training	72	317
Boston Housing	Kaggle	training	15	333
Brfss	github.com	training	34	245
Canberra Observations	others	training	13	19,918
Casualties 2015	data.gov.uk	test	16	186,189
Census Income KDD	UCI ML	test	42	199,523
CleanEHR	others	training	62	1,979
Cylinder Bands	UCI ML	training	40	540
EDF Stocks	github.com	test	7	5,425
Elnino	UCI ML	test	9	782
FACAMemberList2015	github.com	training	21	72,220
French Fries	github.com	training	10	696
Fuel	github.com	training	35	941
Geoplaces2	UCI ML	training	20	130
HES	ukdataservice.ac.uk	training	65	4,600
Housing Price	Kaggle	training	81	1,460
Intel Lab	others	test	8	1,048,576
Inspection Outcomes	others	test	22	1,477
MINY Vendor	data.gov	test	18	897
Pedestrian	others	training	9	37,700
Phm	others	training	16	75,814
Processed Cleveland	UCI ML	training	14	303
Rodents	others	training	39	35,549
Sandy Related	NYC OpenData	training	38	87,444
SFR55_2017_national_characteristics	gov.uk	test	41	735
SFR55_2017_one_plus_sessions	gov.uk	test	31	228,282
Survey	others	test	27	1,259
Tao	github.com	training	7	736
Tb	github.com	training	23	5,769
TundraTraits	github.com	training	17	73,428
User Profile	UCI ML	training	19	138
Vehicles 2015	data.gov.uk	test	23	257,845
4 csv files	mass.gov	test	27 (avg.)	46,934 (avg.)
9 csv files	data.gov	test	14 (avg.)	3904 (avg.)
2015ReportedTaserData	github.com	training	69	610
16 csv files	T2Dv2 Gold Standard	test	5 (avg.)	127 (avg.)
83492acc-1aa2-4e80-ad05-28741e06e530.csv	github.com	training	15	886

Table A.1: Information about the data sets used.

A.3 Derivations for the Training

The task is to update the parameters of the PFSMs, given a set of columns X and their column types \mathbf{t} . Since the columns are assumed to be independent, the gradient can be calculated by summing the gradient of each column. In the interest of simplicity, here we only derive the gradient for a column \mathbf{x} of type k . We would like to maximize the posterior probability of the correct type k given a data column \mathbf{x} , which can be rewritten as follows:

$$\log p(t = k|\mathbf{x}) = \underbrace{\log p(t = k, \mathbf{x})}_{L_c} - \underbrace{\log p(\mathbf{x})}_{L_f}. \quad (\text{A.1})$$

We now present the derivations of the gradients w.r.t. the transition parameters where $\theta_{q,\alpha,q'}^\tau$ denotes the transition parameter from state q to q' emitting the symbol α in the τ^{th} PFSM. Note that $\tau \in \{1, \dots, K\}$ where K is the number of PFSMs.

We now differentiate these two terms, in section (4.1) and (4.2) respectively.

A.3.1 Derivative of L_c

$$\begin{aligned}
\frac{\partial L_c}{\partial \theta_{q,\alpha,q'}^\tau} &= \frac{\partial \log p(t = k, \mathbf{x})}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \frac{\partial \left(\log p(t = k) \prod_{i=1}^N p(x_i | t = k) \right)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \frac{\partial \left(\log p(t = k) + \sum_{i=1}^N \log p(x_i | t = k) \right)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{i=1}^N \frac{\partial \log p(x_i | t = k)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{i=1}^N \frac{1}{p(x_i | t = k)} \frac{\partial p(x_i | t = k)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{i=1}^N \frac{1}{p(x_i | t = k)} \frac{\partial \left(\sum_{z'} p(z_i = z', x_i | t = k) \right)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{i=1}^N \frac{1}{p(x_i | t = k)} \frac{\sum_{z'} \partial p(z_i = z', x_i | t = k)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{i=1}^N \frac{1}{p(x_i | t = k)} \frac{\sum_{z'} \partial \left(p(z_i = z' | t = k) p(x_i | z_i = z') \right)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{i=1}^N \frac{1}{p(x_i | t = k)} \frac{\partial \left(\pi_k^k p(x_i | z_i = k) + \pi_k^m p(x_i | z_i = m) + \pi_k^a p(x_i | z_i = a) \right)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{i=1}^N \frac{1}{p(x_i | t = k)} \frac{\pi_k^k \partial p(x_i | z_i = k)}{\partial \theta_{q,\alpha,q'}^\tau}. \tag{A.2}
\end{aligned}$$

When τ is not equal to k , eq. (2) becomes 0. On the other hand, if $\tau = k$, then we would need to calculate $\frac{\partial p(x_i | z_i = \tau)}{\partial \theta_{q,\alpha,q'}^\tau}$, where $p(x_i | z_i = \tau)$ can be rewritten as $\sum_{q_{0:L}} p(x_i, q_{0:L} | z_i = \tau)$ as x_i is generated by a PFSM. Note that $q_{0:L}$ denotes the states visited to generate x_i .

The derivative can be derived as follows:

$$\begin{aligned}
\frac{\partial p(x_i|z_i = \tau)}{\partial \theta_{q,\alpha,q'}^\tau} &= \frac{\partial \left(\sum_{q_{0:L}} p(x_i, q_{0:L}|z_i = \tau) \right)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{q_{0:L}} \frac{\partial p(x_i, q_{0:L}|z_i = \tau)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{q_{0:L}} p(x_i, q_{0:L}|z_i = \tau) \frac{\partial \log p(x_i, q_{0:L}|z_i = \tau)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{q_{0:L}} p(x_i, q_{0:L}|z_i = \tau) \frac{\partial \log \left[T^\tau(q_0) \left(\prod_{l=0}^{L-1} T^\tau(q_l, x_i^l, q_{l+1}) \right) F^\tau(q_L) \right]}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{q_{0:L}} p(x_i, q_{0:L}|z_i = \tau) \frac{\partial \sum_{l=0}^{L-1} \left(\log T^\tau(q_l, x_i^l, q_{l+1}) \right)}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{q_{0:L}} p(x_i, q_{0:L}|z_i = \tau) \sum_{l=0}^{L-1} \frac{\partial \log T^\tau(q_l, x_i^l, q_{l+1})}{\partial \theta_{q,\alpha,q'}^\tau}, \\
&= \sum_{q_{0:L}} p(x_i, q_{0:L}|z_i = \tau) \sum_{l=0}^{L-1} \frac{\delta(q_l, q) \delta(x_i^l, \alpha) \delta(q_{l+1}, q')}{T^\tau(q_l, x_i^l, q_{l+1})}, \\
&= \sum_{q_{0:L}} \sum_{l=0}^{L-1} p(x_i, q_{0:L}|z_i = \tau) \left(\frac{\delta(q_l, q) \delta(x_i^l, \alpha) \delta(q_{l+1}, q')}{T^\tau(q_l, x_i^l, q_{l+1})} \right), \\
&= \sum_{q_{0:L}} \sum_{l=0}^{L-1} p(q_l = q, q_{l+1} = q', q_{0:L \setminus \{l, l+1\}}, x_i|z_i = \tau) \left(\frac{\delta(x_i^l, \alpha)}{T^\tau(q, x_i^l, q')} \right), \\
&= \sum_{l=0}^{L-1} \sum_{q_{0:L}} p(q_l = q, q_{l+1} = q', q_{0:L \setminus \{l, l+1\}}, x_i|z_i = \tau) \left(\frac{\delta(x_i^l, \alpha)}{T^\tau(q, x_i^l, q')} \right), \\
&= \sum_{l=0}^{L-1} \frac{\delta(x_i^l, \alpha) p(q_l = q, q_{l+1} = q', x_i|z_i = \tau)}{T^\tau(q, x_i^l, q')}. \tag{A.3}
\end{aligned}$$

Hence, we need to evaluate the joint probability $p(q_l = q, q_{l+1} = q', x_i|z_i = \tau)$ for each l where $x_i^l = \alpha$, which can be found by marginalizing out the variables $q_{0:L \setminus \{l, l+1\}}$:

$$p(q_l = q, q_{l+1} = q', x_i|z_i = \tau) = \sum_{q_{l'}} p(q_l = q, q_{l+1} = q', q_{l'}, x_i|z_i = \tau), \tag{A.4}$$

where l' denotes $\{0 : L\} \setminus \{l, l+1\}$. This can be calculated iteratively via Forward-Backward Algorithm where the forward and backward messages are defined iteratively

as follows:

$$\begin{aligned} v_{l \rightarrow l+1}(q_l) &= \sum_{q_{l-1}} T^\tau(q_{l-1}, x_i^l, q_l) v_{l-1 \rightarrow l}(q_{l-1}), \\ \lambda_{l+1 \rightarrow l}(q_{l+1}) &= \sum_{q_{l+2}} T^\tau(q_{l+1}, x_i^{l+2}, q_{l+2}) \lambda_{l+2 \rightarrow l+1}(q_{l+2}), \end{aligned} \quad (\text{A.5})$$

We can then rewrite $p(q_l = q, q_{l+1} = q', x_i | z_i = \tau)$ as follows:

$$p(q_l, q_{l+1}, x_i | z_i = \tau) = (v_{l \rightarrow l+1}(q_l) \bullet \lambda_{l+1 \rightarrow l}(q_{l+1})) \odot T^\tau(q_l, x_i^{l+1}, q_{l+1}), \quad (\text{A.6})$$

where \bullet and \odot denote respectively outer and element-wise product.

A.3.2 Derivative of L_f

Let us now take the derivative of the second term L_f :

$$\begin{aligned} \frac{\partial L_f}{\partial \theta_{q, \alpha, q'}^\tau} &= \frac{\partial \log p(x)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \frac{\partial \sum_{i=1}^N \log p(x_i)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \sum_{i=1}^N \frac{\partial \log p(x_i)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \sum_{i=1}^N \frac{1}{p(x_i)} \frac{\partial \left(\sum_{t'} \sum_{z'} p(t = t', z_i = z', x_i) \right)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \sum_{i=1}^N \frac{1}{p(x_i)} \frac{\partial \left(\sum_{z'} p(t = \tau, z_i = z', x_i) \right)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \sum_{i=1}^N \frac{1}{p(x_i)} \frac{\partial \left(\sum_{z'} p(t = \tau) p(z_i = z' | t = \tau) p(x_i | z_i = z') \right)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \sum_{i=1}^N \frac{1}{p(x_i)} \frac{p(t = \tau) \partial \left(\pi_\tau^\tau p(x_i | z_i = \tau) + \pi_\tau^m p(x_i | z_i = m) + \pi_\tau^a p(x_i | z_i = a) \right)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \sum_{i=1}^N \frac{1}{p(x_i)} \frac{p(t = \tau) \pi_\tau^\tau \partial p(x_i | z_i = \tau)}{\partial \theta_{q, \alpha, q'}^\tau}, \\ &= \sum_{i=1}^N \frac{p(t = \tau) \pi_\tau^\tau \partial p(x_i | z_i = \tau)}{p(x_i) \partial \theta_{q, \alpha, q'}^\tau}. \end{aligned} \quad (\text{A.7})$$

Let us now put all the equations together. When we are calculating the derivative of eq. (1) w.r.t. the correct machine, i.e. $\tau = k$, the derivative becomes the following:

$$\begin{aligned}
\frac{\partial \log p(t = k|x)}{\partial \theta_{q,\alpha,q'}^\tau} &= \sum_{i=1}^N \left(\frac{\pi_k^k}{p(x_i|t = k)} \frac{\partial p(x_i|z_i = k)}{\partial \theta_{q,\alpha,q'}^k} - \frac{\pi_k^k p(t = k)}{p(x_i)} \frac{\partial p(x_i|z_i = k)}{\partial \theta_{q,\alpha,q'}^k} \right), \\
&= \sum_{i=1}^N \left(\pi_k^k \frac{\partial p(x_i|z_i = k)}{\partial \theta_{q,\alpha,q'}^k} \left(\frac{1}{p(x_i|t = k)} - \frac{p(t = k)}{p(x_i)} \right) \right), \\
&= \sum_{i=1}^N \frac{\pi_k^k}{p(x_i|t = k)} \frac{\partial p(x_i|z_i = k)}{\partial \theta_{q,\alpha,q'}^k} \left(1 - \frac{p(t = k)p(x_i|t = k)}{p(x_i)} \right), \\
&= \sum_{i=1}^N \frac{\pi_k^k}{p(x_i|t = k)} \frac{\partial p(x_i|z_i = k)}{\partial \theta_{q,\alpha,q'}^k} \left(1 - \frac{p(t = k, x_i)}{\sum_{k'} p(t = k', x_i)} \right). \tag{A.8}
\end{aligned}$$

When we are calculating the derivative of eq. (1) w.r.t. the wrong machines, i.e. $\tau \neq k$ this becomes:

$$\frac{\partial \log p(t = k|x)}{\partial \theta_{q,\alpha,q'}^\tau} = - \sum_{i=1}^N \left(\frac{\pi_\tau^\tau p(t = \tau)}{p(x_i)} \frac{\partial p(x_i|z_i = \tau)}{\partial \theta_{q,\alpha,q'}^\tau} \right). \tag{A.9}$$

Lastly, we ensure the parameters remain positive and normalized using the softmax function. We define $T_\tau(q, \alpha, q') = \exp T_\tau^z(q, \alpha, q') / (\exp F_\tau^z(q) + \sum_{\alpha', q''} \exp T_\tau^z(q, \alpha', q''))$ and $I_\tau^z(q) = \exp I_\tau^z(q) / \sum_{q'} \exp I_\tau^z(q')$. We now update these new unconstrained parameters using the new gradient calculated via the chain rule: $\partial f / \partial T_\tau^z(q, \alpha, q') = (\partial f / \partial T_\tau(q, \alpha, q')) (\partial T_\tau(q, \alpha, q') / \partial T_\tau^z(q, \alpha, q'))$.

A.4 The Outputs of the PADS Library

We have mentioned previously that the outputs generated by the PADS library do not directly address our problem. We present a sample from an example test dataset in Table A.2, and a part of the corresponding output of the PADS library.

The outputs are interpreted starting from the bottom. In this case, the data is defined as an array of “struct” type named Struct_194. This is further characterized as a combination of various “union” types. For example, let us consider the first one named Union_19 which consists of a constant string ` `, another constant string `year`, and an integer type. However, this can be more complicated as in type Union_165 consisting of two struct types Struct_192 and Struct_164. Note that the former is further divided into a union type, whereas the latter is described as a combination of some constant strings and a float type. As the reader can see, it can become difficult and time-consuming to interpret an output. Moreover, the output becomes more complex

year	winner/2nd	NULL	scores	total	money (us\$)
1998	fred couples	1	64-70-66-66-66	332	414000
 	bruce lietzke	2	65-65-71-62-69	332	248400
1997	john cook	1	66-69-67-62-63	327	270000
 	mark calcavecchia	2	64-67-66-64-67	328	162000
1996	mark brooks	1	66-68-69-67-67	337	234000
 	john houston	2	69-71-65-65-68	338	140400
1995	kenny perry	1	63-71-64-67-70	335	216000
 	david duval	2	67-68-65-67-69	336	129600
1994	scott hoch	1	66-62-70-66-70	334	198000
 	fuzzy zoeller	t2	70-67-66-68-66	337	82133.34
 	lennie clements	t2	67-69-61-72-68	337	82133.33
 	jim gallagher jr.	t2	66-67-74-62-68	337	82133.33
1993	tom kite	1	67-67-64-65-62	325	198000

Table A.2: A sample test dataset.

when delimiters are inferred correctly, as this can prevent the types from column specific.

```

#include "vanilla.p"
Punion Union_19 {
    v_stringconst_12 Pfrom("&nbsp;");
    "year";
    Puint16 v_inrange_4;
};
.
.
.
Punion Union_189 {
    v_stringconst_173 Pfrom("money (us$)");
    Puint32 v_intconst_169 : v_intconst_169 == 72600;
};
Pstruct Struct_192 {
    '\';
    Union_189 v_union_189;
    '\';
};
Pstruct Struct_164 {
    '\';
    Pfloat64 v_float_156;
    '\';
};
Punion Union_165 {
    Struct_192 v_struct_192;
    Struct_164 v_struct_164;
};
Precord Pstruct Struct_194 {
    '\';
    Union_19 v_union_19;
    "\, ";
    Union_62 v_union_62;
    ",\''";
    Union_86 v_union_86;
    "\, ";
    Union_121 v_union_121;
    ',';
    Union_141 v_union_141;
    ',';
    Union_165 v_union_165;
};
Psource Parray entries_t {
    Struct_194[];
};

```

Figure A.1: A fragment of the PADS output for a given dataset.

A.5 Scalability of the Methods

Table A.3 denotes the number of rows, columns, unique elements, and the time passed to infer column types.

dataset	# cols	# rows	U	hypoparsr	messytables	ptype	readr	TDDA	Trifacta
21329809_0_...	6	156	699	2.595	0.001	0.011	0.010	0.0001	1.333
24036779_0_...	7	83	490	4.836	0.001	0.011	0.015	0.0001	1.143
24142265_0_...	6	100	271	2.345	0.001	0.007	0.002	0.0001	1.333
26270372_1_...	5	19	71	1.007	0.001	0.004	0.002	0.0001	2.200
28086084_0_...	6	224	229	1.475	0.002	0.006	0.002	0.0001	1.333
28154036_0_...	5	10	39	2.199	0.001	0.003	0.004	0.0001	1.400
28646774_0_...	6	8	45	1.613	0.001	0.004	0.002	0.0001	1.333
29886325_0_...	6	254	1063	3.572	0.002	0.015	0.014	0.0001	1.667
34899692_0_...	4	92	321	0.978	0.001	0.009	0.001	0.00004	2.250
40534006_0_...	4	39	118	2.667	0.001	0.004	0.002	0.00004	2.000
41480166_0_...	6	224	229	1.343	0.001	0.005	0.002	0.0001	1.500
44005578_0_...	4	8	31	2.014	0.001	0.003	0.001	0.00004	2.000
44206774_0_...	6	89	279	2.505	0.031	0.006	0.002	0.0001	1.500
47709681_0_...	4	408	706	3.432	0.006	0.021	0.002	0.00004	2.250
78891639_0_...	6	202	862	2.927	0.002	0.013	0.002	0.0001	1.333
8468806_0_...	7	110	489	3.654	0.001	0.007	0.002	0.0001	1.143
accident2016	18	555	1835	10.022	0.005	0.009	0.006	0.0001	0.611
accidents_2015	32	140056	609343	-	0.895	1.635	0.045	0.0003	0.281
auto	26	205	911	3.557	0.001	0.011	0.006	0.0002	0.308
casualties_2015	16	186189	140158	-	1.191	0.858	0.030	0.0002	0.625
census_inc...	42	199523	102028	-	0.853	0.423	0.054	0.0004	0.238
data_gov_10012_1	14	145	779	4.023	0.003	0.012	0.011	0.0002	0.571
data_gov_10151_1	21	99	775	2.227	0.002	0.006	0.002	0.0002	0.429
data_gov_12252_1	5	258	756	2.734	0.002	0.015	0.002	0.00005	1.600
data_gov_16834_1	24	15055	2911	-	0.056	0.035	0.005	0.0002	0.333
data_gov_18386_1	16	1238	3975	17.705	0.007	0.024	0.002	0.0001	0.500
data_gov_323_1	6	13260	11740	94.726	0.065	0.157	0.006	0.00005	1.500
data_gov_3397_1	18	437	534	5.785	0.002	0.005	0.002	0.0002	0.500
data_gov_356_1	8	1279	6993	25.147	0.008	0.068	0.003	0.0001	1.000
data_gov_5134_1	10	3366	12198	293.392	0.023	0.110	0.007	0.0001	0.700
edf_stocks	7	5425	4702	23.288	0.049	0.063	0.005	0.0001	1.143
elnino	9	782	1032	12.59	0.005	0.012	0.002	0.0001	1.220
inspection_ou...	22	1477	3115	12.349	0.007	0.014	0.002	0.0002	0.364
intel_lab	8	1048576	112912	-	8.956	2.183	0.266	0.0001	1.250
mass_1	12	131316	803207	-	0.703	12.502	0.150	0.0001	0.750
mass_2	19	44990	12452	-	0.275	0.137	0.015	0.0002	0.474
mass_5	53	8282	72659	-	0.029	0.115	0.009	0.001	0.189
mass_6	23	3148	3363	36.681	0.012	0.018	0.002	0.0002	0.348
miny_vendor	18	897	6728	7.148	0.006	0.030	0.002	0.0002	0.500
school_char...	41	735	16832	3.873	0.005	0.034	0.001	0.0003	0.220
school_sessions	31	228282	78913	-	1.010	0.514	0.039	0.0003	0.290
survey	27	1259	1622	7.036	0.008	0.015	0.001	0.0003	0.296
vehicles_2015	23	257845	141278	-	1.641	0.710	0.034	0.0002	0.391

Table A.3: Size of the test datasets and the times in seconds it takes to infer column types per column (on average), where U denotes the number of unique data entries in a dataset.

Appendix B

Appendix for ptype-cat

We share the available datasets, their sources (e.g., Kaggle and OpenML), the text sequences extracted from their meta-data and our annotations for data types and categorical values at <https://bit.ly/2Ra2Vu7>. Here, we briefly describe these data sets, and report the number of datasets and the counts of data types in each nested cross-validation fold, respectively in Tables B.1 and B.2.

- Abalone: measurements of physical properties of Abalones used to predict their ages.
- Ada Prior: a dataset extracted from census data to discover high revenue people.
- Analcatdata Broadwaymult: information about Broadway shows such as their types (e.g., play and musical), ratings from 0 to 5, and the number of Tony nominations and awards earned.
- Analcatdata Homerun: a collection of game-by-game statistics about home runs of baseball players.
- Anneal: measurements about physical and chemical properties of products after annealing, which is a heating process to reduce their hardness, is applied.
- Atari Head: a collection of actions and eye movements while playing Atari video games.
- Australian: information about credit card applications and whether they are approved or rejected.
- Auto MPG: technical specifications of cars.

- Autos: information about cars such as their characteristics, insurance risk ratings and normalized losses in use as compared to other cars.
- Backache: information about women and back pain levels they experience during pregnancy.
- Bank Marketing: a dataset about marketing campaigns of a Portuguese banking institution used to predict if the client will subscribe a term deposit.
- Banknote Authentication: a dataset of banknote images used to predict whether a banknote is genuine or forged.
- Breast Cancer Wisconsin Diagnostic (WDBC): a dataset about the characteristics of the cell nuclei present in the image of a fine needle aspirate of a breast mass.
- Chscase Geysers: a dataset of interval times between eruptions of the Old Faithful Geysers at Yellowstone National Park, measured during August 1978 and 1979.
- CleanEHR: see Appendix A.2 for the description of this dataset.
- Click Prediction Small: a dataset about advertisements shown alongside search results, and whether or not people clicked on these ads.
- Colleges AAUP: information on faculty salaries for 1161 American colleges and universities.
- Consolidation Centres: descriptions of construction sites that would possibly use the services of a Construction Consolidation Center (CCC).
- Contraceptive Method Choice (CMC): a subset of the 1987 National Indonesia Contraceptive Prevalence Survey, which is used to predict the current contraceptive method choice of a woman based on her demographic and socio-economic characteristics.
- Cover Type: information about wilderness areas in northern Colorado used to predict forest cover type from cartographic variables.
- Cylinder Bands: information about process delays known as cylinder bands in rotogravure printing.

- Diabetes (scikit-learn): information about patients such as their characteristics (e.g., age and sex) and various health-related measurements (e.g., average blood pressure and their disease progression).
- Echocardiogram-UCI: a dataset about patients' heart attacks and health conditions, which is used to predict whether a patient will survive at least one year.
- Electricity Prices ICON: a dataset collected from a cloud computing service to predict the price of its electricity consumption.
- Emotions: a dataset about songs and associated emotions such as happy-pleased and relaxing calm.
- Eucalyptus: information about the suitability of seed lots for soil conservation in seasonally dry hill country.
- Forest Fires: a meteorological dataset used to predict the burned area of forest fires in the northeast region of Portugal.
- Fri-C2-1000-50 and Fri-C4-250-25: two artificially generated datasets using the Friedman functions with varying colinearity degrees, numbers of samples and features.
- Fruitfly: a dataset collected from male fruitflies to analyze the effect of sexual activity on their lifespans.
- Geoplaces2: see Appendix A.2 for the description of this dataset.
- Grub-damage: information about grass grub population and the loss (both pasture damage and economic loss) caused by grass grubs.
- Hypothyroid: thyroid disease records supplied by the Garavan Institute.
- Image: a benchmark dataset that consists of 2000 natural scene images, where 135 features are extracted for each image, and the corresponding labels such as desert, mountains, sea, sunset and trees.
- Iris: the famous UCI dataset containing measurements about plants.
- Jannis: a collection of images used to label the regions in an image, where the labels are Animals, Man-made objects, Persons and Landscape.

- JM1 and MC1: two collections of statistics extracted from source codes to predict whether or not the corresponding software causes a defect.
- Jungle Chess - Lion vs. Elephant: a dataset consisting of features extracted from the game called “Jungle Chess” and the outcome of the games played between two pieces of lion and elephant.
- KDD Cup 99: a subsample of the data from the 1999 ACM KDD Cup where the task was to build a network intrusion detector.
- KEGG Metabolic Reaction Network: a graph dataset where product compounds are considered as node, and genes are treated as edges.
- Kick: a dataset about cars used to predict if a car purchased at an auction is a bad buy, i.e., the car has serious issues that prevent it from being sold to customers.
- King-Rook vs. King-Pawn (Kr-vs-kp): an UCI dataset about chess end-games, where the 36 features describe various positions on the board and the 37th feature describes whether the white can win or not.
- LED: an extension of the UCI LED display dataset, where decimal digits are display with 24 features rather than 7 features.
- Localization Data for Person Activity (LDPA): localization recordings of five people performing different activities, where each person wears four sensors (ankle left, ankle right, belt and chest).
- Magic Telescope: a dataset generated via a Monte Carlo simulation that simulates registration of gamma particles in a ground-based telescope.
- Midwest Survey Nominal: a dataset that contains individual responses from surveys about regional identification.
- MIP 2016 PAR10 Classification: PAR10 performances of modern solvers on the solvable instances of MIPLIB 2010.
- MofN-3-7-10: a dataset consisting of 10 binary features used to evaluate “M-of-N” rules.
- Molecular Biology Promoters: E. coli gene sequences used to recognize promoters in strings that represent nucleotides (one of A, G, T or C).

- **Nomao**: a dataset about places (e.g., name, phone and localization) used for deduplication.
- **NYC-home-parks**: information about the manufactured home parks in New York City such as their names, addresses and counties in which they are located.
- **Ozone Level 8hr**: a ground ozone level dataset collected from 1998 to 2004 at the Houston, Galveston and Brazoria area.
- **Page Blocks**: a collection of the page layouts of documents annotated with labels such as text, horizontal line, picture, vertical line and graphic.
- **Parkinson Speech Dataset with Multiple Types of Sound Recordings**: a dataset consisting of features extracted from audio recordings (e.g., numbers and short sentences pronounced by people with parkinson's disease and healthy people) and a rating that denotes how severe a person's condition is.
- **Pasture**: information about areas such as their biophysical properties and how they are managed (fertilizer application/stocking rate) which is used to predict pasture production.
- **Pharynx**: information about patients with squamous carcinoma and the treatment methods applied (either radiation therapy alone or radiation therapy with together with a chemotherapeutic agent).
- **Philippine**: features about cells of zebrafish embryo and manually annotated labels that denote whether they are in division (meiosis) or not.
- **Poker Hand**: information about poker hands each of which consists of five playing cards drawn from a standard deck of 52.
- **Premier League Odds and Prob**: a dataset that contains the probabilities generated with the statistical models and matches odds for each Premier League match in the 2014-2015 season.
- **Rabe 176**: information about schools (e.g., facilities and faculty credentials) and student achievements.
- **Ringnorm**: an implementation of Leo Breiman's ringnorm example (Breiman, 1996), that consists of 2 classes where each class is drawn from a multivariate normal distribution.

- River Ice: a dataset about estimated river ice (length) fraction based on images from USGS Landsat satellite missions.
- Rodents: see Appendix A.2 for the description of this dataset.
- Satellite: a modified version of satellite observations which are used to classify an image into the soil category of the observed region.
- Scene: a collection of features extracted from scene images and the corresponding labels such as “Urban” and “Not Urban”.
- Sick: a modified version of thyroid disease records supplied by the Garavan Institute.
- Sleuth Case 1202: information (e.g., age, sex, education, work experience and salary) about employees of a bank that was sued for sex discrimination.
- Slump: a multivariate regression dataset used to predict three properties of concrete (slump, flow and compressive strength) as a function of the content of seven concrete ingredients, e.g., cement, fly ash, blast furnace slag, water, superplasticizer, coarse aggregate and fine aggregate.
- SPECT: a dataset consisting of binary features extracted from cardiac images and the corresponding image labels (e.g., normal and abnormal), used to generate diagnoses rules.
- Squash Stored: measurements (e.g., weight, sweetness and flavour) about squash fruits that are transported by refrigerated cargo vessels, used for quality evaluation.
- Student Alcohol Consumption: information (social behaviors, gender and study-related variables such as weekly study time and grades) about students in secondary school.
- TAE: information about teaching assistants (TAs), courses and their teaching performance categorized into “low”, “medium” and “high”.
- Thoracic Surgery: a thoracic surgery dataset collected from patients who underwent major lung resections for primary lung cancer in the years 2007-2011.

- **USP05**: a dataset about university student software projects used to estimate the efforts in hours required to complete a project.
- **Vinnie**: a collection of statistics about the shooting performance of Vinnie Johnson of the Detroit Pistons during the 1985-1986.
- **Volcanoes-b6** and **Volcanoes-d4**: two datasets with varying focus of attention that contain images of volcanoes in Venus collected by the Magellan spacecraft and labels provided by human experts that denote their beliefs over the locations of volcanoes within the images.
- **Wall Robot Navigation**: measurements collected by using ultrasound sensors from a robot navigating through a room.
- **Weather**: a synthetically generated dataset about the suitability of weather conditions for playing an unspecified game.
- **White Clover**: measurements about white clovers, used to analyze how high temperatures in summer impact white clover population.
- **Wholesale Customers**: a dataset about clients of a wholesale distributor such as their annual spending on diverse product categories.
- **Wikipedia Adventure**: a dataset obtained from a survey of Wikipedia editors who played a gamified tutorial.
- **Womens' Clothing Reviews**: an e-commerce dataset containing information about clothing products and their reviews by customers.
- **Young People Survey**: a dataset obtained from a survey about young people and their preferences/habits.
- **Zoo**: a dataset that contains 17 binary attributes describing animals and a class attribute.

Phase	Outer Fold	Inner Fold	# Datasets	Column Type					Total
				Categorical	Date	Float	Integer	String	
Training	1	1	54	580	19	1258	210	52	2119
Validation	1	1	14	201	20	40	121	8	390
Training	1	2	54	711	31	926	311	52	2031
Validation	1	2	14	70	8	372	20	8	478
Training	1	3	54	565	34	1091	246	34	1970
Validation	1	3	14	216	5	207	85	26	539
Training	1	4	55	686	35	685	255	48	1709
Validation	1	4	13	95	4	613	76	12	800
Training	1	5	55	582	37	1232	302	54	2207
Validation	1	5	13	199	2	66	29	6	302
Training	1	-	68	781	39	1298	331	60	2509
Test	1	-	18	120	9	166	180	5	480
Training	2	1	55	587	24	1187	220	54	2072
Validation	2	1	14	99	0	156	163	3	421
Training	2	2	55	610	11	1096	353	47	2117
Validation	2	2	14	76	13	247	30	10	376
Training	2	3	55	478	19	1082	300	31	1910
Validation	2	3	14	208	5	261	83	26	583
Training	2	4	55	582	20	730	305	45	1682
Validation	2	4	14	104	4	613	78	12	811
Training	2	5	56	487	22	1277	354	51	2191
Validation	2	5	13	199	2	66	29	6	302
Training	2	-	69	686	24	1343	383	57	2493
Test	2	-	17	215	24	121	128	8	496

Table B.1: Summaries per fold.

Phase	Outer Fold	Inner Fold	# Datasets	Column Type					Total
				Categorical	Date	Float	Integer	String	
Training	3	1	55	663	40	987	272	48	2010
Validation	3	1	14	99	0	156	163	3	421
Training	3	2	55	604	11	1106	366	44	2131
Validation	3	2	14	158	29	37	69	7	300
Training	3	3	55	560	35	872	339	28	1834
Validation	3	3	14	202	5	271	96	23	597
Training	3	4	55	658	36	530	357	39	1620
Validation	3	4	14	104	4	613	78	12	811
Training	3	5	56	563	38	1077	406	45	2129
Validation	3	5	13	199	2	66	29	6	302
Training	3	-	69	762	40	1143	435	51	2431
Test	3	-	17	139	8	321	76	14	558
Training	4	1	55	610	43	543	261	30	1487
Validation	4	1	14	99	0	156	163	3	421
Training	4	2	55	551	14	662	355	26	1608
Validation	4	2	14	158	29	37	69	7	300
Training	4	3	55	605	35	510	338	22	1510
Validation	4	3	14	104	8	189	86	11	398
Training	4	4	55	560	39	448	347	27	1421
Validation	4	4	14	149	4	251	77	6	487
Training	4	5	56	510	41	633	395	27	1606
Validation	4	5	13	199	2	66	29	6	302
Training	4	-	69	709	43	699	424	33	1908
Test	4	-	17	192	5	765	87	32	1081
Training	5	1	55	567	46	1217	308	56	2194
Validation	5	1	14	99	0	156	163	3	421
Training	5	2	55	508	17	1336	402	52	2315
Validation	5	2	14	158	29	37	69	7	300
Training	5	3	55	562	38	1184	385	48	2217
Validation	5	3	14	104	8	189	86	11	398
Training	5	4	55	433	41	1059	399	33	1965
Validation	5	4	14	233	5	314	72	26	650
Training	5	5	56	594	42	696	390	47	1769
Validation	5	5	13	72	4	677	81	12	846
Training	5	-	69	666	46	1373	471	59	2615
Test	5	-	17	235	2	91	40	6	374

Table B.2: Summaries per fold.

Appendix C

Appendix for ptype-semantics

Below, we give additional information about our knowledge graph of units (Appendix C.1), describe the datasets used (Appendix C.2), present regular expressions used to parse unit symbols (Appendix C.3) and present the derivations for inference in our model (Appendix C.4).

C.1 Additional Information about Our Knowledge Graph

We support the following dimensions: acceleration, amount of substance, angle, area, capacitance, catalytic activity, charge, currency, current, data storage, data transfer rate, dimensionless, dynamic viscosity, electric potential, electrical conductance, electrical resistance, energy, flux density, force, frequency, illuminance, inductance, instance frequency, irradiance, kinematic viscosity, length, linear mass density, luminance, luminous flux, luminous intensity, magnetic field, magnetic flux, magnetomotive force, mass, mass flow, power, pressure, radiation absorbed dose, radiation exposure, radioactivity, sound level, speed, temperature, time, torque, typographical element, volume, volume (lumber), volumetric flow.

To query WikiData, Wikipedia and QUDT, we respectively use wikidata (Minhee, 2017), wikipedia (Goldsmith, 2016) and pyqudt (Brown, 2019).

C.2 Brief Description of the Datasets

The datasets used can be briefly described as follows:

- Arabica, Robusta: a collection of reviews about coffee beans.

- HES (Household Electricity Survey): time series measurements of the electricity use of domestic appliances (to gain access to the data, please register at <https://tinyurl.com/ybbqu3n3>).
- Huffman: the Huffman Prairie flight trials in 1904, which is available through the U.S. Centennial of Flight Commission.
- Maize Meal: a list of maize meal products.
- MBA: a list of products in a grocery shop.
- Open Units: a list of 1,000 standard servings of branded drinks and their alcohol content.
- PHM Collection: information on the collection of Powerhouse Museum Sydney, including textual descriptions, physical, temporal, and spatial data as well as, where possible, thumbnail images.
- 143...6, 143...23 and 228...96: a set of data Web tables from T2Dv2 Gold Standard to evaluate matching systems on the task of matching Web tables to the DBpedia knowledge base.
- query_2, query_4: a set of tables extracted from WikiData using the properties of height or weight.
- Zomato: information about restaurants extracted from Zomato.

C.3 Regular Expressions for Parsing Unit Symbols

We use the following regular expression to parse a unit symbol from a given text:

```
import re
numeric_with_string_const_pattern = r"""
    [-+]? # optional sign
    (
        (?: \d+ \/ \d+ ) # 1/4 etc
        |
        (?: \d* [.,] \d+ ) # .1 .12 .123 etc 9.1 etc 98.1 etc
        |
        (?: \d+ \.? ) # 1. 12. 123. etc 1 12 123 etc
    )
```

```

) ?
# whitespace as separator
(?: [\s]*) ?

# followed by optional characters (alphanumeric,
# whitespace and some punctuation marks)
([\w\s.!?\\-]*) ?
"""
rx = re.compile(numeric_with_string_const_pattern, re.VERBOSE)

```

C.4 Derivations for Inference

C.4.1 Column Type

The posterior distribution of column type t can be derived as follows:

$$\begin{aligned}
p(t = k|\mathbf{x}) &\propto p(t = k, \mathbf{x}), \\
&= p(t = k) \prod_{i=1}^N p(x_i|t = k), \\
&= p(t = k) \prod_{i=1}^N \left[\sum_{u_i=1}^{L_k} \sum_{z_i \in \{u_i, m, a\}} p(x_i, z_i, u_i|t = k) \right], \\
&= p(t = k) \prod_{i=1}^N \left[\sum_{u_i=1}^{L_k} p(u_i|t = k) \left(w_{u_i}^k p(x_i|z_i = u_i) \right. \right. \\
&\quad \left. \left. + w_{u_i}^m p(x_i|z_i = m) + w_{u_i}^a p(x_i|z_i = a) \right) \right].
\end{aligned}$$

C.4.2 Row Label

Let us assume that $t = k$ according to $p(t|\mathbf{x})$, the posterior distribution of column type.

Then we can write the posterior distribution of row label z_i given $t = k$ and x_i as:

$$p(z_i = j|t = k, x_i) = \frac{p(z_i = j, x_i|t = k)}{\sum_{\ell \in \{u_i, m, a\}} p(z_i = \ell, x_i|t = k)},$$

where $p(z_i = j, x_i | t = k)$ can be calculated as follows:

$$\begin{aligned} p(z_i = j, x_i | t = k) &= \sum_{l=1}^{L_k} p(u_i = l, z_i = j, x_i | t = k), \\ &= \sum_{l=1}^{L_k} p(u_i = l | t = k) w_l^j p(x_i | z_i = j). \end{aligned} \quad (\text{C.0})$$

C.4.3 Row Unit

The posterior probabilities of each row unit u_i given $t = k$, $z_i = j$ and x_i can be written as:

$$p(u_i = l | t = k, z_i = j, x_i) = \frac{p(u_i = l, x_i | t = k, z_i = j)}{\sum_{u_i=1}^{L_k} p(u_i = l, x_i | t = k, z_i = j)},$$

where $p(u_i = l, x_i | t = k, z_i = j)$ can be calculated as follows:

$$\begin{aligned} &= p(u_i = l | t = k) p(x_i | t = k, u_i = l, z_i = j), \\ &= p(u_i = l | t = k) p(x_i | z_i = j). \end{aligned}$$

C.4.4 Column Unit

$$\begin{aligned} p(u_i = l | t = k, x_i) &\propto p(u_i = l | t = k) p(x_i | u_i = l, t = k), \\ &= p(u_i = l | t = k) \left[\sum_{\ell \in \{u_i, m, a\}} \left(p(x_i, z_i = \ell | u_i = l, t = k) \right) \right], \\ &= p(u_i = l | t = k) \left[\sum_{\ell \in \{u_i, m, a\}} \left(p(z_i = \ell | u_i = l) p(x_i | z_i = \ell) \right) \right]. \end{aligned}$$

Bibliography

- Agresti, A. (2003). *Categorical data analysis*, volume 482. John Wiley & Sons, 3 edition.
- Bahl, L., Brown, P., De Souza, P., and Mercer, R. (1986). Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 49–52. IEEE.
- Breiman, L. (1996). Bias, variance, and arcing classifiers. Technical report, Statistics Department, University of California, Berkeley.
- Brown, G. (2019). pyqudt. <https://pypi.org/project/pyqudt/> [Accessed on 06/02/2020].
- Brown, P. F. (1987). *The Acoustic-Modeling Problem in Automatic Speech Recognition*. Ph.D. Dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.
- Castelijns, L. A., Maas, Y., and Vanschoren, J. (2020). The ABC of data: A classifying framework for data readiness. In Cellier, P. and Driessens, K., editors, *Machine Learning and Knowledge Discovery in Databases*, volume 1167 of *Communications in Computer and Information Science*, pages 3–16. Springer.
- Ceritli, T., Williams, C. K. I., and Geddes, J. (2020). ptype: probabilistic type inference. *Data Mining and Knowledge Discovery*, 34(3):870—904.
- Chambers, C. and Erwig, M. (2010). Reasoning about spreadsheets with labels and dimensions. *Journal of Visual Languages & Computing*, 21(5):249–262.
- Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15.

- Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data minning guide. pages 10–29.
- Chen, J., Jiménez-Ruiz, E., Horrocks, I., and Sutton, C. (2019a). Colnet: Embedding the semantics of web tables for column type prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 29–36.
- Chen, J., Jimenez-Ruiz, E., Horrocks, I., and Sutton, C. (2019b). Learning semantic annotations for tabular data. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*.
- Dasu, T. and Johnson, T. (2003). Exploratory data mining and data cleaning: An overview. In *Exploratory Data Mining and Data Cleaning*, volume 479, chapter 1, pages 1–16. John Wiley & Sons, Inc., New York, NY, USA, 1 edition.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.
- Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.
- Döhmen, T., Mühleisen, H., and Boncz, P. (2017). Multi-Hypothesis CSV Parsing. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*.
- Dupont, P., Denis, F., and Esposito, Y. (2005). Links between probabilistic automata and hidden Markov models: Probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371.
- Eduardo, S., Nazábal, A., Williams, C. K. I., and Sutton, C. (2020). Robust variational autoencoders for outlier detection and repair of mixed-type data. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4056–4066. PMLR.

- Edwards, A. L. (1948). Note on the "correction for continuity" in testing the significance of the difference between correlated proportions. *Psychometrika*, 13(3):185–187.
- Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics.
- Fisher, K. and Gruber, R. (2005). PADS: A domain-specific language for processing ad hoc data. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI'05)*, volume 40(6), page 421–434. Association for Computing Machinery.
- Fisher, K., Walker, D., Zhu, K. Q., and White, P. (2008). From dirt to shovels: Fully automatic tool generation from ad hoc data. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, volume 43, pages 421–434. Association for Computing Machinery.
- Foppiano, L., Romary, L., Ishii, M., and Tanifuji, M. (2019). Automatic identification and normalisation of physical measurements in scientific literature. In *Proceedings of the ACM Symposium on Document Engineering*, page 24. Association for Computing Machinery.
- Gill, A. (1962). The basic model. In *Introduction to the Theory of Finite-State Machines*, pages 1–15. McGraw-Hill Book Company.
- Goldsmith, J. (2016). wikipedia. <https://pypi.org/project/wikipedia/> [Accessed on 06/02/2020].
- Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610.
- Grecco, H. E. (2019). pint Documentation Release 0.10.dev0. <https://buildmedia.readthedocs.org/media/pdf/pint/latest/pint.pdf> [Accessed on 05/08/2019].

- Guo, P. J., Kandel, S., Hellerstein, J. M., and Heer, J. (2011). 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*, pages 65–74. Association for Computing Machinery.
- Hand, D. J., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining*. MIT Press.
- Hernández-Lobato, J. M., Lloyd, J. R., Hernández-Lobato, D., and Ghahramani, Z. (2014). Learning the semantics of discrete random variables: Ordinal or categorical. In *NeurIPS Workshop on Learning Semantics*.
- Hignette, G., Buche, P., Dibia-Barthélemy, J., and Haemmerlé, O. (2009). Fuzzy annotation of web data tables driven by a domain ontology. In *European Semantic Web Conference*, pages 638–653. Springer Berlin Heidelberg.
- Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). Introduction to automata theory, languages, and computation, 2nd edition. *ACM SIGACT News*, 32(1):60–65.
- Jiang, H. (2010). Discriminative training of HMMs for automatic speech recognition: A survey. *Computer Speech & Language*, 24(4):589–608.
- Kandel, S., Paepcke, A., Hellerstein, J., and Heer, J. (2011). Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. Association for Computing Machinery.
- Keil, J. M. and Schindler, S. (2018). Comparison and evaluation of ontologies for units of measurement. *Semantic Web*, 10:1–19.
- Kittler, J., Hatef, M., Duin, R. P. W., and Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239.
- Lagi, M. (2016). Quantulum. <https://github.com/marcolagi/quantulum> [Accessed on 05/08/2019].
- Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR.
- Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710.

- Limaye, G., Sarawagi, S., and Chakrabarti, S. (2010). Annotating and searching web tables using entities, types and relationships. *Proceedings of the 36th International Conference on Very Large Data Bases*, 3(1-2):1338–1347.
- Lindenberg, F. (2017). messytables Documentation Release 0.3. <https://media.readthedocs.org/pdf/messytables/latest/messytables.pdf> [Accessed on 29/06/2018].
- Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pre-training approach. *arXiv preprint arXiv:1907.11692*.
- Majoor, A. and Vanschoren, J. (2018). Auto-cleaning dirty data: the data encoding bot. Technical report, Technical University of Eindhoven. https://github.com/openml/ARFF-tools/blob/master/Data_Encoding_bot_Report.pdf [Accessed on 06/08/2020].
- McDaniel, G. (1994). *IBM Dictionary of Computing*, page 21. McGraw-Hill, Inc. <https://www.ibm.com/ibm/history/documents/pdf/glossary.pdf> [Accessed on 11/06/2020].
- Minhee, H. (2017). Wikidata. <https://pypi.org/project/wikipedia/> [Accessed on 06/02/2020].
- Nádas, A., Nahamoo, D., and Picheny, M. A. (1988). On a model-robust training method for speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(9):1432–1436.
- Nazábal, A., Williams, C. K. I., Colavizza, G., Smith, C. R., and Williams, A. (2020). Data engineering for data analytics: A classification of the issues, and case studies. *arXiv preprint arXiv:2004.12929*.
- Paz, A. (1971). *Introduction to Probabilistic Automata*, volume 78. Academic Press, Inc., New York, NY, USA.
- Pearson, R. K. (2006). The problem of disguised missing data. *ACM SIGKDD Explorations*, 8(1):83–92.
- Petricek, T., Guerra, G., and Syme, D. (2016). Types from data: Making structured data first-class citizens in F#. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*.

- Qahtan, A. A., Elmagarmid, A., Castro Fernandez, R., Ouzzani, M., and Tang, N. (2018). FAHES: A robust disguised missing values detector. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2100–2109. Association for Computing Machinery.
- Quinn, J. A., Williams, C. K. I., and McIntosh, N. (2009). Factorial switching linear dynamical systems applied to physiological condition monitoring. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(9):1537–1551.
- Rabin, M. O. (1963). Probabilistic automata. *Information and Control*, 6(3):230–245.
- Rabin, M. O. and Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.
- Raman, V. and Hellerstein, J. M. (2001). Potter’s wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 381–390. Morgan Kaufmann Publishers Inc.
- Rijgersberg, H., Van Assem, M., and Top, J. (2013). Ontology of units of measure and related concepts. *Semantic Web*, 4(1):3–13.
- Samadian, S., McManus, B., and Wilkinson, M. D. (2014). Automatic detection and resolution of measurement-unit conflicts in aggregated data. *BMC Medical Genomics*, 7(1):S12.
- Shbita, B., Rajendran, A., Pujara, J., and Knoblock, C. A. (2019). Parsing, representing and transforming units of measure. *Modeling the World’s Systems*.
- Stochastic Solutions (2018). Test-Driven Data Analysis. <https://tdda.readthedocs.io/en/tdda-1.0.23/constraints.html> [Accessed on 08/04/2019].
- Trifacta (2018). Trifacta Wrangler. <https://www.trifacta.com/> [Accessed on 27/06/2018].
- Valera, I. and Ghahramani, Z. (2017). Automatic discovery of the statistical types of variables in a dataset. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3521–3529. PMLR.

- Van Assem, M., Rijgersberg, H., Wigham, M., and Top, J. (2010). Converting and annotating quantitative data tables. In *International Semantic Web Conference*, pages 16–31. Springer Berlin Heidelberg.
- Vergari, A., Molina, A., Peharz, R., Ghahramani, Z., Kersting, K., and Velera, I. (2019). Automatic Bayesian density analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., and Carrasco, R. C. (2005). Probabilistic finite-state machines - Part I. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1013–1025.
- Vrandečić, D. and Krötzsch, M. (2014). Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85. <https://query.wikidata.org/> [Accessed on 25/11/2019].
- Wickham, H. and Grolemund, G. (2016). *R for data science: Import, tidy, transform, visualize, and model data*, chapter 8, pages 137–138. O’Reilly Media, Inc., 1 edition. <http://r4ds.had.co.nz/data-import.html> [Accessed on 24/07/2018].
- Wickham, H., Hester, J., Francois, R., Jylänki, J., and Jørgensen, M. (2017). readr 1.1.1. <https://cran.r-project.org/web/packages/readr/readr.pdf> [Accessed on 29/06/2018].
- Williams, C. K. I. and Hinton, G. E. (1991). Mean field networks that learn to discriminate temporally distorted strings. In *Proceedings of the 1990 Connectionist Models Summer School*, pages 18–22. Morgan Kaufmann Publishers, Inc.
- Williams, J., Negreanu, C., Gordon, A. D., and Sarkar, A. (2020). Understanding and inferring units in spreadsheets. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 1–9.
- Wolfram|Alpha (2019). Units Overview-Wolfram Language Documentation. <https://reference.wolfram.com/language/tutorial/UnitsOverview.html> [Accessed on 05/08/2019].
- Zwacklbauer, S., Einsiedler, C., Granitzer, M., and Seifert, C. (2013). Towards disambiguating web tables. In *International Semantic Web Conference (Posters & Demos)*, pages 205–208.