KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Learning Symbolic Latent Representations for Relational Data

Sebastijan Dumančić

Supervisor:
Prof. dr. ir. Hendrik Blockeel

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

December 2018

Learning Symbolic Latent Representations for Relational Data

Sebastijan DUMANČIĆ

Examination committee:
Prof. dr. Carlo Vandecasteele, chair
Prof. dr. ir. Hendrik Blockeel, supervisor
Prof. dr. ir. Jesse Davis
Prof. dr. ir. Johan Suykens
Prof. dr. David Poole
(University of British Columbia, Canada)
Dr. Mathias Niepert
(NEC Labs Europe, Germany)
Prof. dr. Sebastian Riedel
(University College London, United Kingdom)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

December 2018

© 2018 KU Leuven – Faculty of Engineering Science
Uitgegeven in eigen beheer, Sebastijan Dumančić, Celestijnenlaan 200A box 2402, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm, electronic or any other means without written permission from the publisher.

Acknowledgements

Looking back at the past four years, I realise choosing to enrol in a PhD was the right choice. Pursuing the PhD has been a wish of mine for a long time, and this experience came with many memories and very few regrets. That is largely due to people that were involved with it, and I want to thank them all.

First and foremost, I would like to thank Hendrik for giving me the opportunity to do research. His advice has helped me to become a better researcher, even though I might have not realised it at the time. He taught me the importance of always keeping an eye on the big picture while still getting every detail right, no matter how small it is. More importantly, he taught me the importance of clear and effective communication. This was a rather hard lesson for me, and Hendrik knows that best: as a starting PhD student I was notoriously uninterested in providing concrete examples for any abstract idea I have had (at the end, no example can cover every aspect of a complex idea, no?). I am sure it took a significant effort just to bare with that. His (sometimes necessarily harsh) feedback during the first years has fortunately taught me better.

I am also grateful to the members of my jury. Discussion with Jesse Davis and Johan Suykens over the past four years have pointed me towards a research idea I was unaware of; these discussions have helped me shape the work in this thesis. I am thankful to Mathias Niepert, David Poole and Sebastian Riedel, not only for agreeing to join my jury: they might not be aware it, but they are the first three non-KU Leuven people that ever showed interest in my research. When I met them in New York, at the StarAI workshop in 2016, I have been struggling to publish my first paper; their genuine interest provided encouragement needed to continue pursuing my ideas. Therefore, it is more than fitting for them to be a part of my jury. Finally, I would like to thank Prof. Carlo Vandecasteele for chairing the public defence, and Prof. Jan Van Humbeeck for chairing the private defence.

I have been lucky to be involved in several collaborations during my PhD. Each

of them taught me a lot. I have enjoyed working on COBRAs with Toon, and was envious of his ability to produce a visualisation of any problem we were facing. It also reminded me to always expect the unexpected – this idea came unexpectedly from simply sitting in the same office, and many solutions we found came from unexpected places. My only regret is that we never managed to move to other snakes. Wannes was always the *guy with solutions*; I am thankful for all of the tips & tricks I've learned from him. I remember well insightful discussions with Tias, not only about the underpinnings of CP, but many unexpected topics that popped up in the meantime. His enthusiasm about, well, everything keeps any discussion vivid. The DeepProbLog journey with Robin, Thomas, Angelika and Luc has taught me a lot about connecting dots that are seemingly difficult to relate and the benefit of having various perspectives on the table. I have enjoyed my time in Heidelberg with Mathias, Alberto, Anja and Daniel. This visit has helped me broaden the perspective on the ideas explored in this thesis, and has also opened new avenues to pursue. Finally, Pedro has shown me how a challenge can make things fun.

I was lucky to pursue my PhD in a group where personal interactions matter. It was a great pleasure to meet all of them, and that on its own has made the experience enjoyable. I am thankful to Jessa for being a true friend, a person to talk to, and for binding the group together. Equally importantly, for introducing me to the secrets of Decadenza and sharing the enthusiasm (as you would expect from a food snob). To Vova, for (astonishingly) various conversations and all the follow-up e-mails about equally diverse information. Especially for the intricacies of a hedgehog behaviour. To Irma, for her positive energy and for being the initiator of many things we enjoyed. To Toon, Elia, Vincent N. and Antoine, for the excellent atmosphere in the office which has stimulated many discussions, work- and fun-related ones equally. I still think this is the best office in the building. To Behrouz, for always showing a different perspective (and for Faloodeh). To Angelika and Guy, for always willing to discuss my ideas. To Pedro, for challenging every opinion (and for squash). To Ondrej and Benjamin, for their grumpiness that gave a special flavour to every discussion. To Sergey, for being so random and to Irina for putting up with him. To Anna, Anton, Antoine, Arcchit, Arne, Bogdan, Evgeniya, Francesco, Guillermo, Jonas, Joris, Kurt, Killian, Kshitij, Leander, Leen, Mohit, Nitesh, Robin, Ruben, Samuel, Siegfried, Stella, Tom, Vaishak, Vincent (the big one), Thanh, Wen-Chi and others for many memorable coffee breaks, Secret Santas, movies and the Escape Room in Ypres.

Some people left the mark in an unexpected (even unintended) way. I am thankful to Nada Lavrač for introducing me to the field SRL during my visit to her lab in Ljubljana; this has convinced me that logic is not as boring as I used to think. To Jan Šnajder, for being an enthusiastic AI professor that initiated my

interest in AI. To Oliver Schulte, for being the first person that ever approached me at a conference (and doing so willingly). To Alex, Ana and Marielle, for the Summer School in Cadiz. To Carolina and Daniel, for my Erasmus year in Leuven.

Summer School of Science has played an important role throughout the years of my PhD. This is probably the most influential experience that persuaded me to start a career in science. I am thankful to people that made it such a special place: Anamarija, Anna-Maria, Jelena, Dora, Dunja, Kristina, Korado, Leonardo, Mario, Marko, Matija, Matilda, Neva, Nikolina, Petras, Renan, and many others.

One regret I have is that I was never particularly good at maintaining old relationships, especially the long-lasting ones from Croatia. But going back to Croatia for Summer and Christmas, I was always looking forward to meeting Viktor, Ivan, Matko and Domagoj. We might not see each other for a while, but I sincerely appreciate that it always feels like we saw each other recently.

A special gratitude goes to my parents, Ljubica and Dražen, for their support over the years. It was not always easy to put up with the idea my brothers and I could come up with, but they have always encouraged us to achieve everything we envisioned. They taught us how to be persistent and not give up when it becomes difficult.

Finally, to Nina. I don't have enough words to thank you. Thank you for entering my life unexpectedly and changing it for the better. For making me the person I am today. For all meows, songs and goofy moments; sushi, curries and Ben 'n' Jerry's; travels, hidings around the apartment and evenings on the couch; Bjorn, Bocko and Robert; for many things I'm forgetting right now. And for making our little oasis in Schipvaartstraat a home.

Sebastijan Dumančić
Leuven, Belgium
November 2018

Abstract

The early 21st century has been largely shaped by the huge amounts of available data generated by the widespread adoption of information technology. This rapid growth of the stored data has created the need for automated tools capable of extracting useful bits from large amounts of data. In turn, such tools have changed our perspective on data from a mere record of an event to a carrier of useful information. Before insights can be drawn from the data, the data has to be first brought into a suitable form by means of *feature engineering* as it can rarely be processed in its *raw* form. This step is usually time- and labour-intensive, and often requires an extensive knowledge of the domain.

In this thesis, we tackle the problem of *representation learning* – how to automate the process of feature construction? We focus on feature construction with rich relational data expressed in form of networks, e.g., biological and traffic networks, as many real-life problems can be easily expressed in this format. To be able to express complex feature over networks, the proposed methods rely on the expressive data representation language of first-order logic.

The first contribution of the thesis is a new versatile relational clustering framework that decouples various sources of relational similarity and combines them in a systematic manner. The second contribution is CUR^{2}LED – a relational representation learning framework that exploits approximate symmetries in relational data as features. The third contribution are *auto-encoding logic programs* – a relational generalisation of auto-encoders, one of the basic representation learning primitives. The fourth contribution is the experimental comparison of various relational representation learning methods that offers insights into the strengths and weaknesses of the existing approaches.

Experiments demonstrate the promise of relational representation learning methods as a building block in any relational machine learning framework as the relational classifiers enhanced with the method introduced in the thesis

improve the performance of relational classifiers.

Beknopte samenvatting

Het begin van de 21ste eeuw wordt gekenmerkt door de grote hoeveelheden beschikbare data die geproduceerd worden door de algemene ingebruikname van informatietechnologieën. Deze snelle groei aan opgeslagen data zorgt voor een nood aan geautomatiseerde toepassingen om nuttige informatie uit de grote hoeveelheden data te onttrekken. Zulke toepassingen hebben op hun beurt ons perspectief op data veranderd van een medium om gebeurtenissen op te slaan tot een drager van nuttige informatie. Voordat inzichten uit data verkregen kunnen worden, moeten de data eerst in een geschikte vorm gebracht worden door middel van feature engineering, aangezien deze slechts zelden in hun ruwe vorm kunnen worden verwerkt. Deze stap is meestal tijds- en arbeidsintensief en vereist vaak een uitgebreide domeinkennis.

In dit proefschrift behandelen we het probleem van het leren van representaties – hoe kan het feature-constructieproces geautomatiseerd worden? We focussen op feature-constructie met rijke relationele data uitgedrukt in de vorm van netwerken, aangezien veel alledaagse problemen gemakkelijk kunnen uitgedrukt worden in dit formaat, bijvoorbeeld biologische netwerken en verkeersnetwerken. Om complexe features van netwerken te kunnen uitdrukken, maken de voorgestelde methodes gebruik van eerst-orde logica als expressieve taal om de data voor te stellen.

De eerste bijdrage van dit proefschrift is een nieuwe veelzijdig framework voor relationele clusteranalyse. Dit framework ontkoppelt verschillende bronnen van similariteit en combineert ze op een systematische manier. De tweede bijdrage is CUR^{2}LED – een framework voor het leren van relationele representaties dat benaderde symmetrieën in de relationele data gebruikt als features. De derde bijdrage zijn auto-encoderende logische programma's – een relationele veralgemening van auto-encoders, een van de basisprimitieven voor relationeel leren. De vierde bijdrage is een experimentele vergelijking tussen verschillende methodes om relationele representaties te leren. Deze vergelijking biedt inzichten in de sterktes en zwaktes van de bestaande manieren.

Experimenten tonen het potentieel aan van methodes voor het leren van relationele representaties als bouwsteen in eender welk framework voor relationeel machinaal leren. Dit potentieel wordt aangetoond doordat de relationele classificatiesystemen uitgebreid met de methodes geïntroduceerd in dit proefschrift de performantie van relationele classificatiesystemen verbeteren.

List of Abbreviations

AI Artificial intelligence. 1–3, 9, 22, 35

AIp Auto-encoding Logic Program. xvi, 94, 95, 97–99, 101, 103–107, 131–133

ANN Artificial neural network. 8, 9, 33, 34, 41, 133

CC Relational Conceptual Clustering. 57, 59, 60, 63–65, 69, 70

CUR^{2}LED Clustering-based Unsupervised Relational Representation Learning with Explicit Distributed Representation. xv–xvii, 74–79, 81, 82, 85–87, 91, 93, 94, 110, 128, 131–134

DL Deep Learning. 5, 94, 95

HS Hybrid similarity. 57, 59, 60, 63–65, 69, 70

HSAG Hybrid similarity on Annotated graphs. 57, 59, 60, 63–65, 69, 70

ILP Inductive logic programming. 6, 9, 22–25, 40, 42, 110, 128

kFOIL kFOIL. 120

KGE Knowledge Graph Embeddings. 38, 39, 75, 81, 109, 110, 115–120, 122–125, 128

LRNN Lifted Relational Neural Network. 40, 41

ML Machine Learning. 2, 4, 8, 9, 20, 31, 32, 36, 37

MLN Markov Logic Network. xv, xvi, 28, 29, 103–106

NT Neighbourhood tree. 78, 80, 87

PILP Probabilistic inductive logic programming. 26

ReCeNT Relational Clustering over Neighbourhood Trees. 55, 59, 60, 62–70, 76, 78

RelNN Relational Neural Network. 40, 41

RIBL Relational Instance-Based Learning. 57–60, 63–65, 69–71

RKOH A rooted kernel for ordered hyper-graphs. 58–60, 63, 64, 69–71

SRL Statistical relational learning. 5, 6, 9, 26, 27, 39, 40, 42, 93–95, 104, 109, 115–117, 119, 123, 124, 131, 133, 134

TILDE Top-Down Induction of Logical Decision Trees. xvii, 82–85, 112

URL Unsupervised Representation Learning. 74, 77

WLST Weisfeiler-Lehman subtree kernel. 58–60, 63–65, 69, 70

Contents

Abstract v

Beknopte samenvatting vii

List of Abbreviations x

Contents xi

List of Figures xv

List of Tables xvii

1 Introduction 1

1.1 Artificial intelligence 1

1.2 Machine learning and logic 2

1.3 Motivations and Problem statement 6

1.4 Thesis contributions 9

1.5 Structure of the thesis 11

2 Learning with logic 15

2.1 Representing data with logic 15

2.1.1 Propositional logic 15

2.1.2 Predicate logic 16
2.1.3 Logic programming 18
2.1.4 Hierarchy of representation languages 20
2.2 Inductive logic programming 22
2.2.1 Learning as Search 23
2.2.2 Language bias 25
2.3 Probabilistic logic programming 26
2.3.1 Problog 27
2.3.2 Markov logic networks 28

3 Learning useful representations of data 31
3.1 Introduction 31
3.2 Representation learning principles 32
3.2.1 Artificial neural networks 33
3.2.2 Auto-encoders 34
3.2.3 Properties of good data representation 35
3.3 Learning representations of relational data 37
3.3.1 Knowledge graph embeddings 38
3.3.2 Capturing data properties with random walks 39
3.3.3 Logic-based approaches 40
3.3.4 Hybrid approaches 43

4 An expressive dissimilarity measure for relational clustering 45
4.1 Introduction 46
4.2 Relational clustering over neighbourhood trees 48
4.2.1 Hyper-graph Representation 48
4.2.2 Neighbourhood tree 50
4.3 Dissimilarity measure 52

4.4 Positioning in the literature 56
4.4.1 Hyper-graph representation 56
4.4.2 Related tasks 57
4.4.3 Relational clustering 57
4.4.4 Complexity analysis 59
4.5 Evaluation 61
4.5.1 Data sets 61
4.5.2 Experimental setup 62
4.5.3 Results 63
4.6 Conclusion 71

5 Learning latent features by capturing approximate symmetries 73
5.1 Introduction 74
5.2 Representation Learning via Clustering 76
5.3 Similarity of Relational Structures 78
5.3.1 Hyper-edge Similarity 78
5.3.2 Similarity Interpretation 80
5.4 Related Work 81
5.5 Experiments and Results 82
5.6 Opening the black box of latent features 86
5.6.1 Discovering the explanations of latent features 87
5.6.2 A few examples 89
5.7 Conclusion 91

6 Auto-encoding logic programs 93
6.1 Introduction 93
6.2 Auto-encoding Logic Programs 95
6.3 Learning as constraint optimisation 98

6.3.1 Search . . . 101
6.3.2 Pre-processing . . . 102
6.4 Experiments and results . . . 103
6.4.1 Results . . . 105
6.5 Conclusion . . . 107

7 Towards a better understanding of relational latent representations 109
7.1 Introduction . . . 110
7.2 Why are CUR^{2}LED representations effective? . . . 110
7.2.1 Experiments and results . . . 111
7.3 On embeddings as an alternative paradigm for relational learning 115
7.3.1 Comparing symbolic and distributional methods . . . 116
7.3.2 Aims, Materials and Methods . . . 117
7.3.3 Comparative results . . . 121
7.3.4 Key messages . . . 124
7.4 Conclusion . . . 128

8 Conclusion 129
8.1 Thesis summary and contributions . . . 129
8.2 Future work . . . 132
8.2.1 Theoretical aspects . . . 132
8.2.2 Practical aspects . . . 133

Bibliography 135

List of Figures

1.1 Usage of the storage technologies 6
1.2 Cognitive ease 7

2.1 Example Problog program 28
2.2 The MLN translation of the Problog program in Figure 2.1 . . . 29

3.1 A choice of problem representation matters 32
3.2 Illustration of an artificial neural networks 33
3.3 Illustration of the auto-encoding principle 35
3.4 Representing relational data with embeddings 38
3.5 Lifted relational neural networks 41
3.6 Relational Neural Networks 41

4.1 Example relational dataset 47
4.2 Representation paradigms of relational data 48
4.3 An illustration of the neighbourhood tree 51
4.4 Semantic decomposition of neighbourhood tree 56

5.1 Learning a feature hierarchy with k-means 75
5.2 An illustration of CUR^{2}LED procedure 79

5.3 Illustration of hyper-edge similarity operators 80
5.4 Discovering the meaning of latent features by analysing their relations 88
5.5 Explanation of the latent features invented on the IMDB dataset 89
5.6 Explanation of the latent features invented for the UWCSE dataset 90

6.1 Illustration of the Auto-encoding logic programs 96
6.2 Performance of the MLN models learned on the original and Alps-induced data representations. 105
6.3 Aspects of learning procedure for Alps 106

7.1 Label entropy of latent representation created by CUR^{2}LED . . 113
7.2 Sparsity of latent representation created by CUR^{2}LED 113
7.3 Contrasting the label entropy and sparsity in latent representation created by CUR^{2}LED 114
7.4 Redundancy in latent representations created by CUR^{2}LED . . 115
7.5 Difference in performance between sub-symbolic and symbolic methods. 121
7.6 Control experiment - results on the filtered version of the WebKB dataset 125

List of Tables

1.1 An example dataset 3

2.1 An example of the multi-instance representation 20

4.1 Aspects of similarity considered by various relational clustering approaches 59

4.2 Complexities of the various methods for relational clustering . . 60

4.3 Characteristics of the datasets used in the relational clustering experiments 61

4.4 Performance of relational clustering approaches 64

4.5 Clustering performance of individual similarity components . . 66

4.6 Affinity aggregation results 68

4.7 Accuracy of the kNN classifier with different (dis)similarity measures. 69

4.8 Runtime comparison of various relational clustering approaches 70

5.1 Predictive accuracy of TILDE models learned on the original and latent representations. 83

5.2 Model selection results with TILDE and CUR^{2}LED 85

5.3 Number of clusters in various $\mathcal{C}$-representations 86

7.1 Properties of the relational classification datasets 119

7.2 Properties of the extracted rules 122

Chapter 1

Introduction

1.1 Artificial intelligence

The development of the early 21st century was largely shaped by the transformative influence of information technology, and on its frontier is Artificial intelligence (AI). Ever since the development of the Google search engine at the beginning of the century, numerous AI algorithms have been used to aid tasks typically done by humans. Its influence is particularly evident in the changing landscape of today's economy. Forbes' Fortune 500 list, ranking the most successful companies worldwide, was typically dominated by oil companies in the 20th century. In recent years, the dominion has shifted towards the IT companies; among the ten most valuable companies, five are IT – Apple, Alphabet, Microsoft, Amazon and Facebook. Although we can say that the AI is the core business only for the Alphabet's subsidiary Google, AI is present in the ever increasing number of products of the other companies.

Despite its importance, it is still difficult to say what AI precisely is. One of the most common definitions describes AI as building *machines that think like humans* [125]. However, this definition is not operative as we still do not understand how the humans think. A more operative definition is that of AI creating *machines that behave intelligently* [141]. This way AI machines only have to act intelligently and it does not require understanding the human intelligence nor the peculiarities of various problem solving skills humans possess. Like human intelligence, AI is a broad field including many different tasks. Examples include problems of planning and scheduling, automated reasoning, theorem proving, robotics, language understanding and many

others. Many problems within AI often require a combination of different methods. The latest two decades have also witnessed many breakthrough applications of AI including DeepBlue [63] which defeated the world chess champion Gary Kasparov, Watson [46] which has defeated the top human players in the question-answering game of Jeopardy!, AlphaGo [130, 131] which has dominated the best human player in the game of Go, self-driving cars and many more.

1.2 Machine learning and logic

One aspect that made AI influential is the growing number of digital devices and the amount of information they produce and store. Extracting useful information from the massive available data has the potential to reveal patterns and insights about many aspects of the everyday life. This is, in a way, similar to the tests designed by psychologists and sociologists interrogating the nature of individuals and society, with a difference that the starting point is not a design of a questionnaire but the available data. This way AI has changed our perspective of data, which has been transformed from a mere record of an event to a carrier of useful information.

Perhaps the most prominent representative of AI nowadays is machine learning (ML), which concerns the extraction of useful patterns from data. Machine learning develops AI systems that *improve their performance with experience*. This means that systems learn concepts and skills from examples – the data provided by the expert, or through interaction with the environment. These systems are therefore not programmed in advance; rather, their skills are acquired from the examples. The impact of such systems is huge: from discovering new knowledge to bringing the power of computer programming to non-experts which could simply show the computer how to solve the task, instead of programming it.

How exploiting such data has influenced the modern world is perhaps best summarised in a quote by the mathematician Clive Humbly:

> Data is the new oil. It's valuable, but if unrefined it cannot really be used. It has to be changed into gas, plastic, chemicals, etc. to create a valuable entity that drives profitable activity; so must data be broken down, analysed for it to have value.

A common idea underlying many machine learning algorithms is that of learning a *model of the world*. This assumes that a learning task is defined

Table 1.1: An example dataset describing pets, their features (columns one to six) and the target concept

ID	Features						Target
	legs	fluffiness	diary	playful	spits?	bites?	
cat	4	4	carnivore	sometimes	no	sometimes	✓
dog	4	5	carnivore	yes	no	no	✓
duck	2	2	herbivore	no	no	no	×
				...			
alpaca	4	5	herbivore	yes	sometimes	no	?

as a system with clearly defined inputs and outputs. Given these inputs and outputs, a task of a typical machine learning algorithm is to discover a *decision function*, a system's gearwheels transforming the inputs to the output(s). Existing machine learning algorithms differ in the gearwheels they use to solve the task, i. e., how they *represent* a decision function.

Example (The art of machine learning in a nutshell) Imagine yourself choosing a new pet. Being an AI enthusiast, and having to choose among a vast variety of potential pets, you decide to use a machine learning algorithm to help you with the decision. The first challenge you are faced with is getting the data. You decide to compile a large database of potential pets and identify their *features* – for example, the number of legs, level of fluffiness of their fur, their diary, whether they are playful of not, and whether they spit or bite. These features define the *inputs* for a machine learning algorithm. We still have to provide the *experience* to the learning algorithm in form of the desired outputs; the input data (the features) and the desired output which define the learning task. The desired output comes from the previous pets you owned and whether you liked having them as pets or not – for example, imagine you have owned a dog and a cat which you liked, and a duck which you were not so fond of. These examples constitute our *training data*, and are collected in a simple Excel sheet (Table 1.1).

Once the data is compiled, you can proceed with learning the mapping from the inputs to the outputs – the decision function. This leaves you with the second choice – what should the decision function look like? One choice could be `IF-THEN` rules:

$$\texttt{IF}\ \text{legs} = 4\ \texttt{AND}\ \text{fluffiness} \geq 4\ \texttt{THEN}\ \text{yes}.$$

An alternative option would be to assign an *importance* to each feature so that a

desirable potential pet gets a positive score, while the un-desirable one gets a negative score:

$$w_1 \cdot \text{legs} + w_2 \cdot \text{fluffiness} + w_3 \cdot \text{diary} + w_4 \cdot \text{playful} - w_5 \cdot \text{spits} - w_6 \cdot \text{bites} - w_7 \geq 0$$

In this case, we would have to replace categorical choices, such as the `playful` feature, with a numerical equivalent. For example, we could replace `no` with 0, `sometimes` with 1 and `yes` with 2. Many choices are possible, and the right one depends on the task at hand.

Once the model has been learned, either by finding the rules or by tuning the weights of the decision function, the model can be applied to make decisions about new examples. For instance, if we wish to know whether an alpaca would make a good pet, we can ask the model. Looking at the first rule, the answer would be `yes` as an alpaca has four legs and a level of fluffiness above 4. This defines a *supervised* machine learning task; if the examples of the output and not provided but have to be discovered, this constitutes an *unsupervised* machine learning task. An example of an unsupervised task is *clustering* which concerns discovering natural groups in data.

As the example demonstrates, it is necessary to differentiate between the representation of *data* and the representation of the *decision function*. This thesis focuses on the data representation aspect and inherits the representation of decision functions from the existing methods.

The choice of data representation involves two issues. First, one has to decide about the data representation *format*. The data format describes the basic form – a template, the data will take. By far the most popular choice is the *tabular representation*, e.g., Excel sheets, but other options include graphs, networks and even programming languages. The second issue is concerned with the *content* of the data itself, i.e., defining the *features* of the task. This populates the template with the content.

Both data representation aspects have a critical impact on the success of the machine learning application. Choosing an insufficient data format prevents us from capturing all intricacies of the data. Even more crucial is the choice of the features. If the features themselves do not contain information, no ML algorithm will be able to successfully solve the task. In contrast, an insufficient data format might not be able to capture *everything* about the data, but it is possible that it captures *most* of the relevant information.

Identifying the right and informative features for the given problem is no easy task. Developing a ML model typically proceeds in four steps: 1) collecting the

data, 2) designing the features, 3) learning the model, and 4) evaluating the model. However, this is rarely a linear procedure and the practitioners typically have to iterate between steps 2 and 4 to find suitable features, which results in a labour-intensive task taking the vast majority of the development [156]. The main reason why finding the informative features is difficult is that one typically uses a machine learning algorithm to solve the problem one does not know how to solve. As the solution is unknown, it is even more difficult to know which information has to be a part of the solution.

Motivated by the difficulty of manually finding informative features, machine learning practitioners have started to develop formal frameworks that can find informative features with little or no expert intervention. These methods fall under umbrella of *representation* or *deep learning* (DL) [55, 5]. *Deep* refers to the idea of creating several layers of *latent features* defined in terms of the given features, before solving the given task. These methods, typically based on artificial neural networks, have sparked a revolution within Artificial Intelligence over the last decade. As a result, many solutions in the fields of computer vision, speech recognition and text processing have witnessed impressive improvements. This problem is the central part of this thesis.

There is the second part of the puzzle that is important for this thesis - data representation format. The vast majority of progress on representation learning has focused on the tabular data format. However, real world data is rarely tabular. *Stack overflow* survey, the yearly survey reporting on the usage of various technologies among the developers all over the world, reports that the most commonly used storage technology are relational databases (Figure 1.1). The progress towards representation learning on structured data, that contain examples and their mutual relationships, has focused on *re-representing* structured data in tabular format. This is clearly limiting as tabular representation can rarely capture all of the intricacies of structured data.

A powerful way to overcome these representational limitations is to use *predicate logic* as a data representation format. The field of AI combining machine learning with logic is known as *Statistical relational learning* (SRL) [54, 114] and *Inductive logic programming* [28]. These methods combine three pillars of Artificial intelligence: it uses *predicate logic* to represent complex data formats, *probability theory* to capture the uncertainty of reasoning and *machine learning* for learning models that leverage both logic and probability from data. This renders SRL methods amongst the most expressive AI methods, i.e., they can easily represent tabular data, networks and graphs as well as entire computer programs.

SRL formalisms have given us powerful tools to represent complex models and

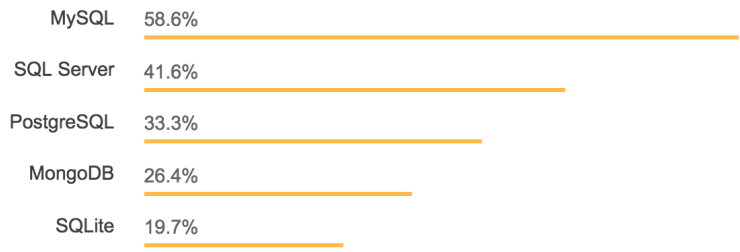


Figure 1.1: Top five most commonly used storage systems, according to *Stack overflow*'s Developer survey 2018

use them for drawing conclusions. However, learning models from scratch has proven to be a difficult task, mostly because it includes expensive inference as a sub-procedure. A lot of progress on learning the rules of such logical models have been made within the ILP community, which ignores the probabilistic aspect and focuses purely on the logical aspect. The key reason why learning is difficult for SRL models is that the learning task can be described as searching for the best predictive formulas. However, the space of all possible formulas is huge and suffers from *combinatorial explosion*.

1.3 Motivations and Problem statement

It is well known in psychology that the ability to solve a task substantially depends on the way the task is represented [68].

Example Consider the tasks in the Figure 1.2. These are two simple addition tasks: one with Arabic and the second one with roman numbers. Both tasks are surely easy to solve. However, paying attention to the effort it takes to solve each of the tasks, you will notice that solving the task on the right requires more effort. Moreover, it will take you more time solve it than the task on the left. Psychologists call this *cognitive ease* [68]; a measure of how easy it is for our brains to process information. The state of cognitive ease is activated by confrontation with familiar concepts which do not require a significant effort. The Arabic system is the system we are confronted with every day and it, thus, does not require any extensive processing effort. In contrast, the Roman system is the one we are rarely confronted with after the primary school education and thus it requires a switch – we are likely to solve it by *mapping* the numbers to the Arabic system. This task is an example of *cognitive strain*.

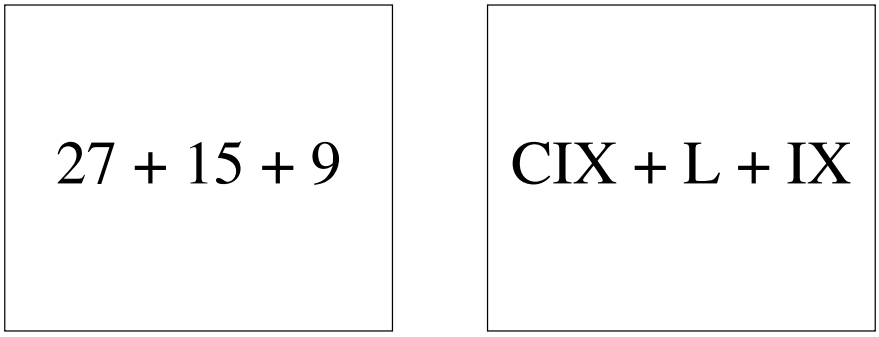


Figure 1.2: **Cognitive ease.** Two tasks illustrate the concept of cognitive ease. The task on the left specifies the problem in the representation familiar to us; the task on the right a representation we can understand but are not used to. Even though both tasks are simple, the right one on average requires more effort in order to be solved.

Example Consider now a typical programming task from the first-year programming course: *given a collection of elements, implement a function that checks whether a given element is present in the collection*. If the collection of elements is small, the choice of the data structure representing the collection does not matter much. However, if the collection of elements is large, for example containing tens of millions of elements, whether a set or an array is used as a data structure makes a big difference. Using the set representation one can check the existence of the elements almost instantly; in contrast, searching through an array results in traversing the entire collection until we find the required element.

Example Many of the objects we interact with on a daily basis are not the *physical manifestations* but *abstractions* that make communication easier. For instance, a *penguin* is an abstract concept, a language construct, referring to a medium-sized bird that does not fly but swims, has a beak, is of black and white colour and lives in places in the southern hemisphere. Moreover, one can say that a bird is the abstraction of a specific group of animals that fly. Such *representation changes* – abstractions, help us to communicate effectively. Without them, communication about categories as specific as a penguin would be tedious as it would require providing the entire definition of a category, not just a single word.

All of the examples above point to the same conclusion – *representation matters*

and *change of representation* is an important tool of intelligence. The first example illustrates how even the simplest task can be made more difficult by using a different representation. The second example illustrates how the change of representation can make a correct but slow solution much more efficient. Finally, the third example illustrates how the introduction of abstract concepts makes communication more effective. Unfortunately, machine learning algorithms were not capable of changing the representation autonomously until very recently. Even recent progress is quite limited in its scope. Therefore, incorporating general methods for representation change into machine learning algorithms has a big potential to substantially increase their problem solving abilities.

The critical limitations of the existing representation learning methods we address in this thesis are the following. First, the vast majority of representation learning approaches are based on the formalism of Artificial neural networks (ANN). ANNs are known to be universal approximators, meaning that they can approximate any desired function given enough data and enough hidden layers, but are focused on tabular and signal-like data, including images, sound and text. They are, thus, not the most suited method for relational data formats. Consequently, the majority of representation learning approaches for relational data focus on re-representing the data in a tabular format, which necessarily introduced a loss of information. Second, ANNs are by their design uninterpretable. Therefore, even though latent representation created by ANNs often perform quite well, it is often impossible to extract knowledge from the novel representation. This undermines the trust in ML models, especially in the domains in which interpretability is not an option but a mandatory requirement. Third, learning representations with ANNs is often *magic* – these methods often have a large number of parameters, and their performance heavily depends on them. Unfortunately, most of the time there is no clear guidance on how to chose these parameters.

To overcome the issues outlined above, we approach the problem of learning representations of relational data from a different perspective: instead of approximating the more expressive logic framework with less expressive tabular format, we will lift the ideas from representation learning with the tabular data to the logical framework. This view offers several benefits. First, it increases the *expressiveness* – what can a model represent?, as it uses a programming language as the representation language. Using a programming language as a representation language allows us to tackle many complicated problems which cannot be expressed in single-table format. Moreover, programming concepts such as recursion allow us to express complicated properties in a compact manner. Second, it is declarative. In contrast to imperative programming paradigms which specify each step towards the

solution, declarative programming paradigms specify the solution and leave it to a solver to find the correct steps. Consequently, programs written in declarative programming languages are typically much shorter than the ones written in the imperative paradigms. Third, it is interpretable. Logical formulas are relatively easy to interpret for humans; that is also true for very large logical systems, if they are written in a *modular* way. This is not true for representation learning approaches based on ANNs, in which representation mapping is achieved through multiplication and addition of often millions of parameters.

Our long-term goal is to develop the tools that allow ML methods, in particular, the ones within the SRL and ILP literature, to change the representation on demand. This will significantly reduce the effort needed from a human expert in order to find informative features for the task. Moreover, as the goal of AI is to develop autonomous *intelligent agents* [125] capable of acting independently in the world, the ability to change the representation for the given task is an important part of it. To contribute to the goal, this thesis explores different ideas approaching the following research question:

> How can we lift the existing representation learning techniques to the rich knowledge representation framework of predicate logic?

Moreover, we will consider representation learning to be a *standalone* procedure. This view of representation learning takes data as an input of the procedure and transform the provided data to a new form, without considering the subsequent learning task. The benefit of this view is that the latent representation of data is obtained only once and is later re-used among many tasks over the same data.

1.4 Thesis contributions

We identify *four main contributions* within this thesis addressing the proposed research question.

A versatile relational clustering framework

The first main contribution of the thesis is a new *versatile relational clustering method* [36]. The proposed clustering framework consists of a novel similarity measure for both objects and relationships in relational data. The main novelty of the proposed framework is that it comprises several *interpretations of similarity* of relational data, e.g., similarity in terms of attributes or structures of the

neighbourhood, in contrast to the existing methods in the literature which typically impose a very strict bias on what the similarity means. The introduced framework specified only the similarity measure and, therefore, can be further combined with any clustering algorithm.

Learning latent features by detecting approximate symmetries

The second main contribution of the thesis is a framework for learning relational latent features by detecting *approximate symmetries* in data, termed CUR^{2}LED [35]. This approach is rooted in the idea that identifying similar objects in data, and finding what makes them similar, is a useful proxy towards identifying good features. To identify the symmetries in objects and their relationships, this approach relies on previously proposed relational clustering methods. As knowing which kind of symmetries would be useful for a task is very difficult, if not impossible, the proposed approach makes use of many kinds of symmetries exploiting the previously introduced notion of interpretation of similarity.

Auto-encoding logic programs

The third main contribution of the thesis are *auto-encoding logic programs* [40, 41] – a logical generalisation of auto-encoders. Auto-encoders are among the most versatile representation learning components applicable to many different learning settings. This versatility makes them very suitable for the variety of SRL learning tasks, such as generative modelling and supervised learning, as the same latent representation can be re-used in many tasks. We introduce a declarative framework for learning relational auto-encoders which are implemented as logic programs, instead of algebraic computation as with more standard auto-encoders.

Analysis of relational representation learning approaches

The fourth contribution of this thesis is the empirical analysis and comparison of several relational representation learning methods. First, we focus on better understanding CUR^{2}LED and show that the created latent features match well with the existing labels, which explains why such latent features are beneficial. Second, we compare symbolic relational learning approaches to the prototypical *knowledge graph embedding methods* – a novel paradigm for relational learning rooted in representation learning, which approaches the

problem by mapping logical symbols to points in Euclidean space. We compare their performance on various relational datasets, identify their strengths and weaknesses and show that a good idea might be to have a hybrid learner exploit the strengths of both sides.

These four contributions serve as an exploration of various ideas in learning relational latent representations. The first contribution introduces a methods that discovers natural structure in data, by grouping similar objects, and the second contribution leverages the discovered structure to define new features. The third contribution pushes this idea forward and defines novel reconstruction-based representation learning framework that entirely depends on logic and logic programs. The fourth contributions takes a critical view of the two relational representation learning paradigms, symbolic and sub-symbolic one, and identifies directions for hybrid learners that leverage the strengths of both paradigms. These advancements contribute to the fundamentals of *symbolic principles* towards relational representation learning, and our hope is that they initiate novel developments that offer a symbolic alternative to the existing methods that transform relational data into vectors that only approximate the rich structure.

1.5 Structure of the thesis

The work in this theses combines the fields of *statistical relational learning* and *representation learning*. We review the basic foundations of Statistical Relational learning in **Chapter 2** and Representation learning in **Chapter 3**.

Chapter 4 presents a versatile clustering framework for relational data. The key contributions of this chapter are (i) a novel similarity measure for relational objects, based on the summarisation of neighbourhood of individual objects, and (ii) a general framework for clustering both objects and relationships in the data. The major novelty of the proposed framework is that it relies on several ways to define a similarity, instead of using only one way to look at it. This chapter is based on the following publication

> DUMANČIĆ, S., AND BLOCKEEL, H. An expressive dissimilarity measure for relational clustering using neighbourhood trees. *Machine Learning 106*, 9-10 (Oct. 2017), 1523–1545

The following chapters explore different ideas towards learning relational latent features. **Chapter 5** introduces the idea of using approximate symmetries in data as a latent representation of data. It relies on the previously introduced

relational clustering framework to identify various approximate symmetries, i.e. groups of relational objects that are *similar* but not *identical* to each other, and treats the identified symmetries as latent features. Furthermore, it introduces a method for explaining the created latent features. **Chapter 6** introduces *auto-encoding logic programs*, a declarative generalisation of the auto-encoder architecture [62] that uses logic programs as a computational framework. It introduces a general modelling framework for auto-encoding logic programs, as well as a general solver based on the constraint satisfaction problem that learns them from the declarative models provided by the user. These chapters are based on the following publications:

> DUMANČIĆ, S., AND BLOCKEEL, H. Clustering-based relational unsupervised representation learning with an explicit distributed representation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17* (2017), pp. 1631–1637
>
> DUMANČIĆ, S., AND BLOCKEEL, H. Demystifying relational latent representations. In *Inductive Logic Programming* (2018), N. Lachiche and C. Vrain, Eds., Springer International Publishing, pp. 63–77. Best Student Paper Award
>
> DUMANČIĆ, S., GUNS, T., MEERT, W., AND BLOCKEEL, H. Auto-encoding logic programs. In *Proceedings of the 2nd Workshop on Neural Abstract Machines and Program Induction* (2018)
>
> DUMANČIĆ, S., MEERT, W., AND BLOCKEEL, H. Theory reconstruction: a representation learning view on predicate invention. In *6th workshop on Statistical Relational Artificial Intelligence StarAI at IJCAI* (2016)
>
> DUMANČIĆ, S., GUNS, T., MEERT, W., AND BLOCKEEL, H. Auto-encoding logic programs. *preparations* (2018)

Chapter 7 summarises the insights found by comparing various relational representation learning methods. The analysis focuses on identifying the strengths and weaknesses of various methods, as well as conditions under which different methods are applicable. This chapter is based on the following publications

> DUMANČIĆ, S., AND BLOCKEEL, H. Demystifying relational latent representations. In *Inductive Logic Programming* (2018), N. Lachiche

and C. Vrain, Eds., Springer International Publishing, pp. 63–77. Best Student Paper Award

DUMANČIĆ, S., GARCIA-DURAN, A., AND NIEPERT, M. On embeddings as alternative paradigm for relational learning. In *Proceedings of the 8th International Worshop on Statistical Relational Artificial intelligence* (2018)

DUMANČIĆ, S., GARCIA-DURAN, A., AND NIEPERT, M. A comparative study of distributional and symbolic paradigms for relational learning. *preparation* (2018)

The final chapter summarises the thesis, discusses its implications and provides a look into directions for *future work*.

Spread throughout the chapters, various intermezzos describe some of the work done during this thesis that does not concern the topic of relational representation learning. These intermezzos describe the work on *semi-supervised active clustering* and *integration of symbolic and neural approaches*, which are published as

VAN CRAENENDONCK, T., DUMANČIĆ, S., AND BLOCKEEL, H. COBRA: A fast and simple method for active clustering with pairwise constraints. In *IJCAI* (2017), ijcai.org, pp. 2871–2877

VAN CRAENENDONCK, T., DUMANČIĆ, S., WOLPUTTE, E. V., AND BLOCKEEL, H. COBRAS: interactive clustering with pairwise queries. In *Symposium on Intelligence Data Analysis (IDA)* (2018), vol. 11191 of *Lecture Notes in Computer Science*, Springer, pp. 353–366

VAN CRAENENDONCK, T., MEERT, W., DUMANČIĆ, S., AND BLOCKEEL, H. COBRASTS: A new approach to semi-supervised clustering of time series. In *Discovery Science* (2018), vol. 11198 of *Lecture Notes in Computer Science*, Springer, pp. 179–193

MANHAEVE, R., DUMANČIĆ, S., KIMMIG, A., DEMEESTER, T., AND DE RAEDT, L. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31* (2018), S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., pp. 3753–3760

Chapter 2

Learning with logic

This chapter briefly introduces the concepts of statistical relational learning. It starts with a short introduction to logic, starting from the propositional logic and gradually climbs the ladder of complexity all the way to the logic programming. It continues with a brief introduction to Inductive logic programming. Finally, it touches upon the contemporary topic of *Probabilistic* Inductive Logic Programming which combines Inductive Logic Programming with probability theory to quantify the uncertainty of the reasoning.

2.1 Representing data with logic

2.1.1 Propositional logic

Propositional logic is a formalism for reasoning about the truth assignments of *propositions*. Propositions are statements about the world being modelled, and can be *true* or *false*.

The propositions are constructed from propositional *variables* and *connectivity operators*. Propositional variables denote *aspects* of the world considered by the model, for instance, *sun*, *rainbow* and *bob_runs*. Variables are used in propositions as *literals* – a propositional variable or its negation. The connectivity operators compose literals into more complicated statements. Such operators are *conjunction* (AND operator), *disjunction* (OR operator), *implication* (IF-THEN operator) and *equivalence* (IF-AND-ONLY-IF operator).

A *logical theory* is a collection of propositions, more precisely, a conjunction of propositions. An *interpretation* is a truth assignment for each of the propositional variables. An interpretation satisfies a given proposition if it evaluates to *true* for the given truth assignments to the variables. An interpretation that satisfies the proposition is a *model* of that proposition.

Example The proposition

$$\texttt{playful} \wedge \texttt{fluffy} \Rightarrow \texttt{ideal\_pet}$$

encodes that when something is playful (variable `playful` is true) and fluffy (variable `fluffy` is true) then it is an ideal pet (variable `ideal_pet` is true). Every interpretation in which `ideal_pet` is true is a model of this proposition.

Once the theory describing the knowledge is available, different *inference* tasks can be considered. One of the fundamental tasks is that of *consistency* or *satisfiability* checking, which checks whether the theory has a model. The task of *validity* checking checks whether every interpretation is a model. The task of *model counting*, a generalisation of both aforementioned tasks, counts the number of models of the theory.

2.1.2 Predicate logic

Propositional logic is a suitable framework for drawing inferences about individual objects or examples (where each example is an interpretation), but encoding more complex knowledge in form of relations among objects is extremely tedious. For instance, to indicate that two animals are of the same species would require introducing new Boolean variables for each pair of animals, such as `same_species_dog_cat`, and specifying their truth assignments.

To overcome this issue, *predicate logic* extends propositional logic with *objects* and *relations* among them, resulting in a powerful language for representing mathematical formalisms. An especially attractive feature of predicate logic is its universal *inference algorithm* – one can state a set of axioms, or facts, and theorems in predicate logic and rely on the *resolution* [119] inference algorithm to derive new axioms and theorems from the old ones.

The language of predicate logic is similar to the one of the propositional logic. The core building blocks of predicate logic formulas are *four* types of symbols: *constant* (referring to individual objects), *variable* (referring to groups of objects), *function* and *predicate* symbols.

The statements in predicate logic, the formulas, are composed of *terms*, *atoms* and *connectivity operators*. A *term* is defined as:

- a constant is a term
- a variable is a term
- if f/n is a function symbol and $t_1, t_2, ..., t_n$ are terms, then function $f(t_1, ..., t_n)$ is a term.

An *atom* is of the form $p(t_1, ..., t_n)$ where p/n is a predicate symbol and $t_1, ..., t_n$ are terms. A *literal* is an atom or its negation.

Finally, a *formula* in predicate logic is defined as:

- An atom is a formula
- `true` and `false` are formulas
- If ϕ and ψ are formulas, so are $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \Rightarrow \psi$, $\phi \Leftrightarrow \psi$ and $\not\phi$
- If ϕ is a formula and X is a variable, so are $\forall X \phi$ and $\exists X \phi$.

A Formula is *ground* if it does not contain logical variables, meaning that the variables are *bound* to the exact objects or individuals.

Evaluating formulas in predicate logic is not as simple as in the propositional logic. The main complication comes from the usage of variables. Thus, in order to specify the interpretation one first has to, for each predicate, map logical variables to a set of domain elements, i.e., each predicate has to be *grounded*. Once groundings are obtained, each grounding of a predicate is associated with `true` or `false`.

Example Mapping the previous example in propositional logic to predicate logic results in the following formula:

$$\forall X \quad \texttt{is\_playful(X)} \wedge \texttt{is\_fluffy(X)} \Rightarrow \texttt{ideal\_pet(X)}.$$

Predicate logic allows us to easily express more complex knowledge relating different objects. For instance, we can refine the previous formula and add a condition that an ideal pet also needs to have at least one friend:

$$\forall X, \exists Y \quad \texttt{is\_playful(X)} \wedge \texttt{is\_fluffy(X)} \wedge \texttt{friends(X,Y)} \Rightarrow \texttt{ideal\_pet(X)}.$$

Assuming the domain {`cat`,`hamster`,`alpaca`}, the groundings of the predicate `is_playful(X)` are: `is_playful(cat)`, `is_playful(hamster)` and `is_playful(alpaca)`.

Predicate logic allows us to represent knowledge about the world in a very compact manner: grounded atoms specify facts about the world, while logical formulas general properties and relationships that would otherwise be too cumbersome to write as facts.

2.1.3 Logic programming

Logic programming uses the language of predicate logic for computer programming. It uses a subset of predicate logic in which the domains are restricted to the *Herbrand base* – the set of all ground atoms that can be constructed from the constant and function symbols in the alphabet. Furthermore, it restricts the formulas to be in the form of *Horn clauses*.

A *clause* is a disjunction of literals:

$$\mathtt{A}_1 \vee \ldots \vee \mathtt{A}_n.$$

More generally, if some of the literals in a clause are *negated*

$$\mathtt{A}_1 \vee \ldots \vee \mathtt{A}_n \vee \neg \mathtt{B}_1 \vee \ldots \vee \neg \mathtt{B}_m$$

then the clause can be written as an *if-then* rule, for instance in the logic programming language *Prolog*:

$$\underbrace{\mathtt{A}_1, \ldots, \mathtt{A}_n}_{\textit{head}} \mathtt{:-} \underbrace{\mathtt{B}_1, \ldots, \mathtt{B}_m}_{\textit{body}}.$$

A comma in the *head* of the rule stands for a disjunction, while in the *body* it indicates the conjunction of literals. A *Horn clause* is a clause with only one positive literal, i.e., $n = 1$.

A very attractive property of clausal logic is that it forms a very general framework for representing knowledge. For example, a clausal statement with $n > 0$ and $m = 0$ indicates a fact. If $n, m > 0$, then such a statement is a rule. A statement with $n = 0$ and $m > 0$ represents a *Prolog query* – a statement whose truth assignment we want to check. Moreover, even the learning algorithm itself can be represented as a set of clauses.

The main difference between predicate logic and logic programming is in the interpretation of the implication operator. Predicate logic ($\Rightarrow$) interprets it in a declarative way, while logic programming takes a procedural interpretation of implication (:-) – meaning that the head of the clause is *true* every time the body is *true*.

Example Some pet owners are pickier than others and require a certain pedigree from their current and future pets. Let us assume that royal ancestry exists among the animal species as well and that some pet owners want royal pets. How do we express this requirement in logic? Let us define royal ancestry as follows: *someone is of royal ancestry if at least one of her ancestors was a king or queen*. Starting simple, some is royal if he or she is a king or queen:

```
royal(X) :- is_king(X).

royal(X) :- is_queen(X).
```

Next, we can extend the rules to the children of kings and queens:

```
royal(X) :- parent(X,Y),is_king(X,Y).

royal(X) :- parent(X,Y),is_queen(X,Y).
```

Specifying these rules so that we could identify royal ancestry in general case is obviously tedious. However, `recursion` allows us to compactly compute all ancestors through the procedural interpretation of implication:

```
ancestor(X,Y) :- parent(X,Y).

ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).
```

The first clause state that X is an ancestor of Y if X is a parent of Y. The second clause states that X is an ancestor of Y is X is a parent of Y who is an ancestor of Y. We can now define the rule for royal ancestry in a simple way:

```
royal(X) :- ancestor(Y,X),is_king(Y).

royal(X) :- ancestor(Y,X),is_queen(Y).
```

Table 2.1: An example of the multi-instance representation

Owner	**Features**				**Target**
	fluffy	playful	spits?	bites?	
Hannah	4	yes	yes	no	✓
	3	no	yes	no	
	5	sometimes	yes	yes	
Steve	1	yes	yes	no	×
	4	yes	no	no	
		...			

2.1.4 Hierarchy of representation languages

Propositional logic and logic programs are two extremes on the spectrum of possible data representations [28]. This spectrum of representations explores a trade-off between the *expressiveness* – the complexity of what can be represented, and *efficiency* – how fast can we perform fundamental inference and learning tasks in a certain representation class. There is an intricate relationship between these properties: expressiveness comes with lower efficiency. Therefore, resorting to a too powerful data representation as a default choice comes at a high computational cost; in contrast, choosing a less but sufficiently expressive representation offers better efficiency without loss of information.

At the bottom of the ladder of expressivity are *Boolean representations* (BR) based on propositional logic. In Boolean representation, each example is associated with a fixed number of features taking values from $\{true, false\}$. This is similar to our pets data (Table 1.1) if all the values are replaced by $true$ and $false$.

One step up the ladder of expressiveness are *multi-valued representations* (MVR). These are similar to the Boolean representations, but each feature can have multiple (> 2) values. This is the exact case as presented in Table 1.1.

One step further are *multi-instance representations* (MIR) [32]. This representation class associates labels with *sets of examples*. Going back to out pet example, let us redefine the ML model to do a different kind of predictions: *do I like the taste in pets of a certain person?* As one person can have multiple pets, multi-instance representation is well suited for this task. An example is given in Table 2.1: an entity *Hannah* owns three pets with different features, and the target information tells us that we like *Hannah*'s taste in pets. The target information is provided on the set of pets, not individual pets. This is obviously a more difficult learning task the decision function being learned is also required to

identify whether a *relevant* information comes from all elements of the set or just a subset.

At the top of the ladder of expressiveness reside *relational representation* (RR) and *logic programs* (LP), with logic programs sitting on the very top. A slight difference between the relational representation and logic programs, which makes their expressiveness differ, is that relational representations require that no functors are used and that the domains are finite.

It is interesting to study this representation classes from the perspective of *reductions*:

> We say that the representation X is *reducible* to a representation Y, denoted as $X \sqsubseteq Y$, there exist two functions $f_e : \mathcal{L}_{eX} \rightarrow \mathcal{L}_{eY}$ mapping the examples e from the representation X to representation Y, and $f_h : \mathcal{L}_{hX} \rightarrow \mathcal{L}_{hY}$ mapping the decision function h from the representation X to representation Y.

An important consequence of this reduction is the following.

> If a problem E is representable by X, and X is reducible to Y, then $f_e(E)$ is representable by Y, where $f_e(E) = \{f_e(e)|e \in E\}$. The the learning problem E in X is solvable using a learning algorithm in Y.

A very interesting theoretical result relating these representations is the following [27]:

> $\text{BR} \sqsubseteq \text{MVR} \sqsubseteq \text{MIR} \sqsubseteq \text{RR} \sqsubseteq \text{LP}$.

Moreover, these reductions are *proper*, meaning that of X reduced to Y then also Y reduces to X. If relational problems can be solved by Boolean representations, why bother with more expressive representation at all?

The reason lies in the expense that comes with reducing a more expressive representation to a less expressive one. Take, for instance, reduction of MVR to BR. To reduce a problem in MVR, say the data in Table 1.1, to the BR, we have to introduce a new variable for each *attribute-value* pair, e.g., *'fluffiness = 4'* becomes a feature. Though the information in the data will be preserved by this reduction, the size of the data representation will increase exponentially. One can also show that every reduction from the higher to lower step of the ladder yields an exponential increase in the data representation size [28]. This clearly demonstrates the strength of relational representation – they can represent

complex data *compactly*, and it is worth keeping the relational representation if the task requires it. Otherwise, reduction to the Boolean or Multi-valued representation could yield such a huge data representations making learning very difficult.

2.2 Inductive logic programming

So far, we have seen why predicate and clausal logic are a desirable framework for data representation. What has not been discussed so far is their connection with machine learning.

Inductive logic programming [28, 85] is the field of AI combining machine learning with logic programming as the representation language. Intuitively, ILP considers the problem of learning a logic theory, a set of clauses, predicting the target predicate.

More specifically, ILP is defined as follows:

> **Given** a set of positive $\mathcal{E}^+$ and negative $\mathcal{E}^-$ examples, $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$, a language bias $\mathcal{L}$ – a language of clauses, a covers relation c, a background knowledge or theory $\mathcal{B}$,
>
> **Find** a set of clauses $\mathcal{T} \subset \mathcal{L}$ such that $\mathcal{T}$ (together with $\mathcal{B}$) covers all positive and no negative examples

Several components form an ILP problem. First, there are positive and negative examples; going back to our *pet* example, positive examples would be pets we previously liked, while the negative examples would be pets we would not like to purchase again. Second, a typical ILP learner requires a specification of how to construct candidate clauses. This is known as *language bias*, and it is usually specified in a form of syntactic constraints on candidate formulas. For instance, one can use only clauses with at most 3 literals in a body and at most 3 distinct logical variables. Third, one has to provide a *covers* relation. Intuitively, *covers* relation specifies how do we use a theory to make predictions about the examples. Typical choice in ILP is to use *entailment* as *covers* relation:

> Let $\mathcal{C}$ be a set of clauses and c be a clause (a fact in our case). $\mathcal{C}$ *logically entails* c, $\mathcal{C} \models c$, if and only if all models of $\mathcal{C}$ are also models of c.

Fourth, *background knowledge* or *theory* represent any additional knowledge one might have about the provided domain. In our pet example, the background

knowledge consists of the information about diet, playfulness and biting habits of potential pets. Additionally, background knowledge can contain additional *rules* encoding experts knowledge, for instance: *an animal that bites is potentially aggressive*.

2.2.1 Learning as Search

Learning ILP models reduces to *searching trough a set of candidate clauses and identifying a few most effective ones*. As the space of all possible formulas would be enormous, searching it blindly would not be feasible. Therefore, we are forced to search the space of possibilities conservatively so that we identify good candidates as fast as possible. Many different ILP algorithms have been proposed so far, but the vast majority of them follow one of the two prominent frameworks of exploring the space of candidate clauses: *general-to-specific* and *specific-to-general*.

To understand these two learning frameworks, we need to understand two notions – *generality* and *specialisation*.

Definition 2.2.1. *(Generality) Let c_1 and $c_2 \in \mathcal{L}$. Clause c_1 is more general than clause c_2, $c_1 \preceq c_2$, if and only if all examples covered by c_2 and also covered by c_1.*

Definition 2.2.2. *(Specialisation) Let c_1 and $c_2 \in \mathcal{L}$. Clause c_2 is specialisation of clause c_1, $c_1 \preceq c_2$, if and only if all examples covered by c_2 and also covered by c_1.*

Algorithm 1: The ILP learning loop

```
Queue ← initialise
Examples ← {...}
Theory ← ∅
while not stop do
    candidate ← take from Queue
    if candidate satisfies quality criteria then
        Theory ← Theory ∪ candidate
        Examples ← Examples \ {c(candidate,Examples)}
    else
        Queue ← Queue ∪ ρ(candidate)
    end if
    Queue ← prune(Queue)
end while
return Theory
```

Both frameworks follow the same steps, illustrated in Algorithm 1, and differ only in small details. They maintain a *queue* of candidates, and iteratively test candidates from the *queue*. If a candidate clause satisfies a certain quality criterion, it is added to the final theory and all examples it *entails* are removed from the example pool. If a candidate does not satisfy the quality criteria, the refinements of the candidate (obtained through the ρ function) are added to the candidate queue. Afterwards, the queue of candidates is pruned by removing unnecessary candidates. This procedure repeats until a certain stop criterion is met.

The two frameworks differ in two main steps: initialisation of the candidate queue, and the ρ function. The *general-to-specific* approaches initialise the queue with an *empty* clause and generate new candidate by adding new literals to a candidate clause, iteratively generating longer and longer clauses. The *specific-to-general* approaches state from very specific clauses, and gradually generalise them to cover more examples.

A wide landscape of ILP algorithms comes from modifying these few choices: queue initialisation, ρ function, stopping and quality criteria. A common choice for the stopping criteria is to stop when no more examples can be covered. The choice for a quality is a more difficult one, and it typically involves specifying the minimal accuracy for a clause to be acceptable, the minimal number of examples to cover and so on.

Example Going back to the example in Table 1.1, assume we are given the following vocabulary: { `legs/2`, `fluffiness/2`, `diary/2`, `playful/2`, `bites/1`, `spits/1` }. We will assume that the predicates with two arguments, such as `legs` and `playful`, take an entity of a pet as the first argument and the second argument states the value of the attribute. The predicate taking only one argument, for instance `bites`, simply state that a pet bites. Searching for a predictive ILP model in a top-down manner will start by exploring all clauses with a single predicate in the body:

```
perfect_pet(X) :- legs(X,2).   perfect_pet(X) :- legs(X,4).
                               ...
 perfect_pet(X) :- spits(X).   perfect_pet(X) :- bites(X).
```

Clauses that were found useful, let us say that is `perfect_pet(X) :- legs(X,4).`, will be extended further:

```
perfect_pet(X) :- legs(X,4),spits(X).  perfect_pet(X) :-
                  legs(X,4),bites(X).

                          ...
```

until a satisfactory model is found.

2.2.2 Language bias

Blindly searching over the space of possible clauses usually generates huge search spaces. To tackle this issue, many ILP system rely on *language bias* [12] to focus the search.

Language bias provides a specification of how the provided predicates can *interact* with each other. This typically includes specifying the following information

Argument types. Given an predicate `b/2` (where `/2` defines the number of arguments the predicate takes), each argument has an associated *type*. For instance, the predicate *friends/2* operates only between entities of type `person`: both arguments are of the type `person`.

Argument roles. Each argument of a predicate has an associated *role*, which specifies how the predicate can be introduced in a clause. Arguments can be of the following types:

- **Input (+):** an input argument needs to be *bounded* to already existing variable in a clause, or any of the variables from the head of the clause
- **Output (-):** an output argument introduces a new variable in the clause
- **Constant (#):** a constant must be introduced in the place of the argument.

For instance, assume you want to extend the clause `h(X,Y) :-` b_1`(X).` with a predicate b_2`/2` having the roles specified as b_2`(+,#)`. Then the first argument of b_2`/2` has to be bounded to one of the existing variables `{X,Y}`, where the constant has to be introduced instead of the second argument (assume c_1). The possible extensions are then:

$$h(X,Y) \texttt{ :- } b_1(X), b_2(X, c_1).$$

$$h(X,Y) \texttt{ :- } b_1(X), b_2(Y, c_1).$$

It the predicate $b_2/2$ would be specified as b_2`(+,-)`, the second argument has to introduce a new variable, e.g.,

```
h(X,Y) :- b1(X),b2(X,Z).
```

Syntactic restrictions. Constraints might be imposed on the syntax of the clauses, for instance

- **Connectivity** specifies which connectivity operators can be used in the bodies of the clauses: conjunction, disjunction and negation. We currently use only the conjunctions and disjunctions.
- **Length bound** limits the lengths of the body of a clause.
- **Term bound** introduces the upper bound on the *maximal* number of terms in a clause.

Given these constraints, we can reduce the search space substantially which can further be pruned through generalisation and specialisation.

2.3 Probabilistic logic programming

Though clausal logic is a powerful representation framework, it still has a drawback when it comes to reasoning: it can only represent things being *true* or *false*, but cannot reason about the uncertainty of conclusion. Therefore, significant work has been devoted towards combining logic reasoning with probability theory, under the name of Statistical relational learning (SRL) and Probabilistic inductive logic programming.

Halpern introduces two formalisms for giving the semantics to systems combining logic an probability [58]. The first formalism assigns probabilities on the domain, i.e., the facts, and is suitable for probabilistic logic involving the uncertain facts. The second formalism assigns probabilities on the possible worlds, and is suitable for giving semantics to formulas describing degrees of belief. In the remainder of the chapter, we describe two prominent approaches within SRL and PILP: *Problog* and *Markov logic networks*.

2.3.1 Problog

Problog [29] is a probabilistic extension of Prolog where some facts can be annotated with the probabilities. It is, therefore, a representative of the first formalism of the Halpern's systematisation.

An example of a Problog program is given in Figure 2.1. The central concept in Problog is a *probabilistic fact*: ground fact f annotated with a probability p (e. g., 0.8::`playful(alpaca)`). *Deterministic facts* are simply the probabilistic facts with $p = 1.0$. Each probabilistic fact corresponds to a Boolean random variable which is *true* with probability p and *false* with probability $1 - p$. Next to the probabilistic facts, Problog consists of deterministic rules equivalent to Prolog clauses (e. g., `cuddly(X) :- fluffiness(X,Y),Y >= 4.`). Probabilistic rules (e. g., 0.3::`bites(X) :- diet(X,carnivore).`) are syntactic sugars which simply introduce a new probabilistic fact reflecting the probability of the rule every time the deterministic rule is `true`. Such probabilistic facts together with rules define the following distribution P_F over the truth assignments to the variables corresponding to probabilistic facts:

$$P_F(F') = \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i). \tag{2.1}$$

The distribution P_F can be then used to defined the *success probability* of a query q – the probability that q (a ground atom) is *true* is a randomly chosen program, as the sum over all programs that entail q:

$$\begin{aligned} P(q) := & \sum_{\substack{F' \subseteq F \\ \exists \theta F' \cup R \models q\theta}} P_F(F') \\ = & \sum_{\substack{F' \subseteq F \\ \exists \theta F' \cup R \models q\theta}} \prod_{f_i \in F'} p_i \cdot \prod_{f_i \in F \setminus F'} (1 - p_i) \, . \end{aligned}$$

SRL systems, such as Problog, are very flexible and can perform various inference tasks, not just the prediction task:

- *success probability:* a query q is given, and the task is to compute $p(q)$
- *marginal probability:* a set of queries Q and evidence E is given, and the task is to compute the marginal probability distribution of each $q \in Q$

$$\texttt{legs(alpaca,4).} \quad 0.9\texttt{::friends(alpaca,otter).}$$

$$\texttt{fluffiness(alpaca,5).} \quad 0.8\texttt{::playful(alpaca).}$$

$$\underbrace{\texttt{diet(alpaca,herbivore).}}_{\textit{deterministic fact}} \quad \underbrace{0.6\texttt{::spits(alpaca).}}_{\textit{probabilistic fact}}$$

$$\texttt{cuddly(X) :- fluffiness(X,Y),Y >= 4.}$$

$$\texttt{ideal\_pet(X) :- cuddly(X).} \quad 0.3\texttt{::bites(X) :- diet(X,carnivore).}$$

$$\underbrace{\texttt{ideal\_pet(X) :- friends(X,otter).}}_{\textit{deterministic rules}} \quad \underbrace{0.2\texttt{::cuddly(X) :- bites(X).}}_{\textit{probabilistic rules}}$$

Figure 2.1: Problog program for the pet decision problem. Probabilistic facts 0.8 ::`playful(alpaca).` states that `alpaca` us playful with certainty of 80%. Deterministic fact `legs(alpaca,4).` states the `alpaca` has four legs with certainty of 100%. A rule `cuddly(X) :- fluffiness(X,Y),Y >= 4.` states that a pet is `cuddly` is its `fluffiness` level is greater or equal to 4.

- *maximum aposteriori:* given a set of queries Q and evidence E, the task is to find the most likely truth-assignment v to the atoms in Q
- *most probable explanation:* the task is to find the most likely world where the given evidence query e holds.

2.3.2 Markov logic networks

Markov logic networks (MLN) [118] fall in the second category of Halpern's classification as they associate probabilities with possible worlds. MLNs start from the view that the formulas in first-order logic can be seen as a set of hard constraints on the set of possible worlds: if a world violates one logical formula, its probability is zero. MLNs aim to soften these constraints by making the worlds that violate the formulas less probable, but not impossible. Each formula thus has an associated weight that specifies how strong the constraint is.

Formally, a Markov logic network is a set of pairs (F_i, w_i), where F_i is a formula written in first-order logic and w_i is a real number reflecting the strength of the constraint. An MLN can be seen as a template for constructing a a Markov

$$\texttt{legs(alpaca,4)} \quad \texttt{friends(alpaca,otter)}$$

$$\texttt{fluffiness(alpaca,5)} \quad \texttt{playful(alpaca)}$$

$$\texttt{diet(alpaca,herbivore)} \quad \texttt{spits(alpaca)}$$

$$\texttt{fluffiness(X,Y),Y} >= 4 \Rightarrow \texttt{cuddly(X)}.$$

$$\texttt{cuddly(X)} \Rightarrow \texttt{ideal\_pet(X)}.$$

$$w_1 \ \texttt{diet(X,carnivore)} \Rightarrow \texttt{bites(X)}$$

$$w_2 \ \texttt{bites(X)} \Rightarrow \texttt{cuddly(X)}$$

$$\texttt{friends(X,otter)} \Rightarrow \texttt{ideal\_pet(X)}.$$

Figure 2.2: MLN model for the pet decision problem.

network [80], given the set of constants C. MLNs defined the probability distribution over possible worlds as

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$$

where $n_(x)$ is the number of true groundings of the formula F_i in x and x represent the truth assignments to atoms appearing in F_i. Similar to Problog, MLNs are capable of answering different types of queries.

Figure 2.2 shows the MLN translation of the pet decision problem introduced before. In contrast to the Problog model, MLNs cannot represent uncertain facts and therefore all facts are deterministic. Uncertainty in MLNs can be handled only through the formulas which can be stochastic (e. g., w_1 `diet(X,carnivore)` $\Rightarrow$ `bites(X)`) with w_1 reflecting the strength of the constraint, and deterministic (e. g., `cuddly(X)` $\Rightarrow$ `ideal_pet(X)`.) with the associated weight being `infinity` (meaning that no possible world can break it).

Chapter 3

Learning useful representations of data

In this chapter, we introduce the basic ideas from representation learning. We identify properties of *good data representations* and illustrate common representation learning approaches. Finally, we summarise the main ideas within the relational representation learning literature.

3.1 Introduction

As we have seen in Section 1.2 of Chapter 1, machine learning intuitively corresponds to *discovering* a decision function f mapping the input data X to some target concept y:

$$f(X) = y. \tag{3.1}$$

This also illustrates why we care so much about the representation of the data: if the relevant information in X is insufficiently captured or represented, whatever f we learn is doomed to be a bad one. As one relies on ML in scenarios when one has little or no knowledge of f, it is very difficult, if not impossible, to know in advance which information is relevant for the mapping. Deploying a ML solution in practice, thus, requires a lot of manual work just to identify the relevant information for the task. Insufficiently captured information does not imply that the relevant information is not present, but instead is not represented

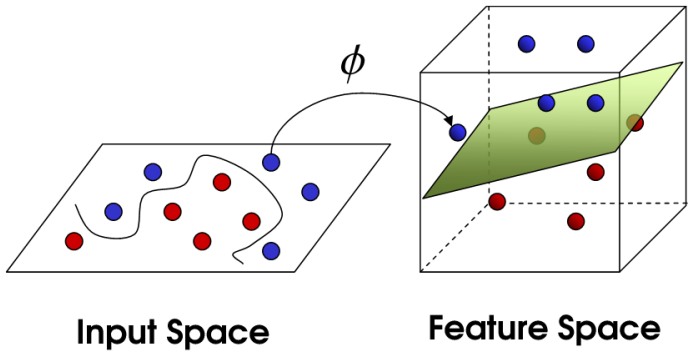


Figure 3.1: A choice of problem representation matters - having a good and informative features can simplify learning [72]

explicitly enough. A typical example is illustrated in Figure 3.1 in which blue dots are difficult to separate from red ones in the original data representation, but transforming the data into a new latent representation makes them easy to separate.

A pleasing property of introducing a representation learning component in a ML pipeline is that is *surpasses the limitation of fixed data representation*. It thus generalises the machine learning equation (Equation 3.1) to:

$$f(m(X)) = y \tag{3.2}$$

where, in addition to the decision functions f, we include the representation mapping function m. Moreover, several mapping function can be *stacked* together to perform multiple representation learning steps.

3.2 Representation learning principles

The core focus of representation learning is on *learning* the function(s) $m_{(i)}$. By the time this thesis was written, several thousands of different representation learning methods exist in the literature [55]. In this section, we illustrate the principles of representation learning on a few prototypical approaches.

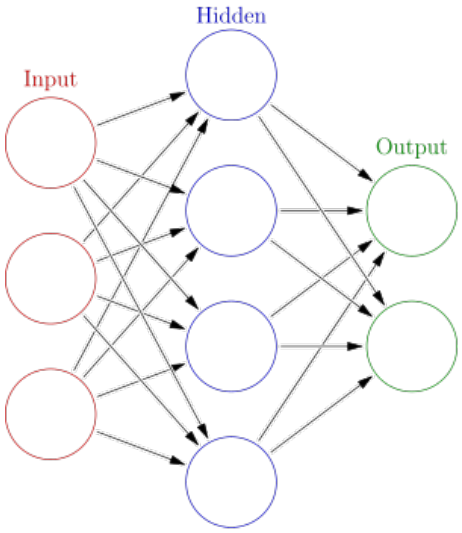


Figure 3.2: An artificial neural network consists of *input neurons* (in red) that represent the data, *output neuron* (green) representing the target concept and *hidden neurons* performing the feature creating step

3.2.1 Artificial neural networks

Representation learning as a field has risen from the advances in *Artificial Neural Networks*. Until this day, almost all representation learning methods are developed within the framework of ANNs.

ANN (Figure 3.2) is a simple mathematical formalisation for machine learning, inspired by the topology of *human brain*. It consists of a *network* of mutually connected *artificial neurons*, also known as *perceptrons* (circles in Figure 3.2). The interactions between neurons are dictated by their connections (edges connecting circles in Figure 3.2).

An ANN in Figure 3.2 consists of three *layers*: an *input layer* which takes the data X, an *output layer* capturing the target concept y and a *hidden layer* performing the representation transformation. ANN operates by activating neurons in different amounts depending on the input data – similar to the human brain. Each neuron in the hidden layer is connected to all of the input neurons but with different *strengths* of connections. In formal terms, the activation of any individual neuron equals

$$a(X) = n(\sum_{i=1}^{N} w_i \times x_i) \tag{3.3}$$

where w_i stands for the strength of the connection, and x_i is the value of an input neuron. The function n is any non-linear function determining the activation based on the weighted sum of the activation of the input neurons, i.e., the data.

The task of learning ANNs is the one of learning each connection's strength given a fixed structure of an ANNs. Once learned, the activation's of the neurons in the hidden layer represents the new representation of data.

This flavour of representation learning belongs to the **supervised** branch, i.e., new representation tailored specifically for the given task (y).

3.2.2 Auto-encoders

A different approach to learning representations are *auto-encoders* [62]. Auto-encoders belong to the **unsupervised** branch of representation learning methods which do not consider a particular supervised or target information when learning representations. A benefit of unsupervised representation learning is that the new representations can be used for many tasks as they are not tailored to a certain task.

Learning new representations in an unsupervised way is substantially more difficult than the supervised case. Given the labelled information, one can verify how useful every feature in the new representation is by, for instance, evaluating to which extent each feature contributes towards distinguishing different target labels. In the unsupervised case, we have no access to such information. The question then is: *What constitutes a good representation if we don't know the target task?*

Auto-encoders introduce a practical way to circumvent tackling this difficult question first. Auto-encoders are ANNs that learn a data representation by trying to compress the original data such that the original data can be reliably reconstructed from the compressed representation. Thus, auto-encoders *leverages* the supervised learning method for unsupervised learning. The two parts of an auto-encoder (Figure 3.3) are an *encoder*, mapping the original data to a new representation, and a *decoder* mapping the latent representation of data back to the original data representation. It is important to note that training auto-encoders does not require any change in learning ANNs, except for the requirement that the ANN reconstructs its own input $y \approx X$.

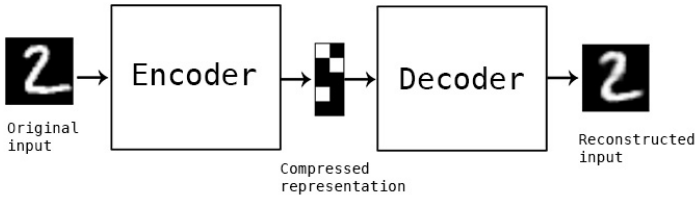


Figure 3.3: **Illustration of the auto-encoding principle.** An auto-encoder *encoding* the provided input image to a *compressed representation* and *decodes* it back to the original feature space

The key in successfully learning a representation with auto-encoders lies in imposing certain constraints on the hidden layer of the auto-encoder. Otherwise, without the constraints, nothing prevents the auto-encoder learning the *identity mapping* which would guarantee the perfect reconstruction. The most common way of preventing the learning of identity mapping is to force the auto-encoders to be *compressive*; this is achieved by restricting the dimension of the hidden layer to be *smaller* than the dimension of the input data. This way, the auto-encoder is forced to exploit patterns in the data in order to maximise the reconstruction from the latent representation.

3.2.3 Properties of good data representation

The success of representation learning has opened many novel research questions, one of the most important ones being: *what constitutes a good data representation?* Bengio [6] argue that explicitly dealing with representations is important and beneficial because representation can be convenient to express general priors about the real worlds. The priors we are interested in should not be task-specific, but likely useful for many different AI tasks. Investigating such general priors has largely shaped the representation learning research over the past decade, and could likely be its most important contribution. Some of the general-purpose representation priors are the following.

Smoothness and locality The most basic representation prior is *smoothness*: learning a function f that is locally consistent, i.e., if $x \approx y$ then $f(x) \approx f(y)$. This property simply states that predictions should be *robust* and generalise to local regions of feature space, reflected in a requirement that similar examples should have a similar prediction.

Multiple explanatory factors The generative data distribution typically comprises several *underlying factors*, and learning something about one of the factors generalises to many configurations of the other factors. This prior gave rise to one of the pillars of representation learning – a *distributed representation*. The concept of distributed representation directly relates the quality of representation to its expressivity: good representations are *expressive*, meaning that a reasonably-sized representation can capture a huge number of possible input configurations. An intuitive way to think about the capacity of the representation is comparing the number of parameters a representation requires to the number of regions it can represent. On the lower end, *one-hot* representations common among many ML methods can represent N regions with N parameters. On the upper end, distributed representations can represent up to 2^N regions with N parameters. Representation learning advertises learning of representations with the latter properties, as the gain in expressivity is evident.

Hierarchical organisation of explanatory factors The knowledge we pose about the world around us is not simply a collection of facts, but a hierarchically organised system. Many concepts are defined in terms of other concepts, structured in a hierarchy so that *low-level concepts* which can be directly observed in the real world constitute the foundations of the hierarchy while the more abstract concepts appear in the higher levels of the hierarchy. The advantage of such hierarchically structured knowledge is that they promote *concept re-use* between the levels of hierarchy, which even further increases the expressivity of the representation. Moreover, concept re-use allows one to express complex concepts *compactly*. This is even more evident considering the *depth* of the hierarchy which indicates the number of levels of abstraction within the hierarchy, i.e., the number of steps from the concept at the bottom of the hierarchy to the most abstract concept. As the number of paths between concepts of different levels of abstractions (i.e. ways to re-use different parts) increases exponentially with the depth, theoretical results have shown that a deep representation can be exponentially more efficient than one that is insufficiently deep [59, 60, 7]

Shared factors across tasks Given many learning tasks of interest, a good data representation contains factors that are shared among many tasks. More precisely, each factor in a representation helps to explain several different tasks. This does not necessarily work for supervised tasks: given data X and a target task Y, a representation useful for explaining the data generating distribution $P(X)$ tend to be useful when learning a predictive task Y given X, $P(Y|X)$. If this property if fulfilled, it allows having a single data representation for

different tasks which can have many practical benefits for big data, e.g., storage. Moreover, it also allows for sharing statistical strength across tasks.

Sparsity Sparsity refers to the notion that for each individual data point only a small subset of factors is relevant. In practical terms, this means that latent representation of each example contains only a small number of *non-zero features*. Sparsity enforces latent representation to partition the instance space in local regions for which only a small number of factors are relevant. This also makes the representation more robust to noise; as only a small number of factors matters, noise is more likely to affect the irrelevant factors.

Natural clustering and manifolds Humans have named categories, they name things, and one of the prevalent hypothesis in neuroscience is that this is due to the statistical regularities in the world we perceive. This means that categories *naturally group* into regions that tend to be well separated and non-overlapping. The regions where different categories lie are therefore *more dense* than the regions between categories; these regions are known as a *data manifold*. Many of the representation learning approaches try to exploit this property.

Simplicity of factor dependencies A good data representation suitable for high-level reasoning should make the final task, e.g. prediction, simple(er) to solve. In practical terms, that means that the dependencies between the factors at the last level of representation hierarchy should be simple and *linear*. From an ML perspective, a final data representation should allow a user to use a simple method, such as linear or logistic regression, to learn a successful predictor.

3.3 Learning representations of relational data

The few examples discussed in the previous section consider learning representation from tabular data only. In this section, we provide a brief overview of the main ideas present in transferring these ideas towards relational data.

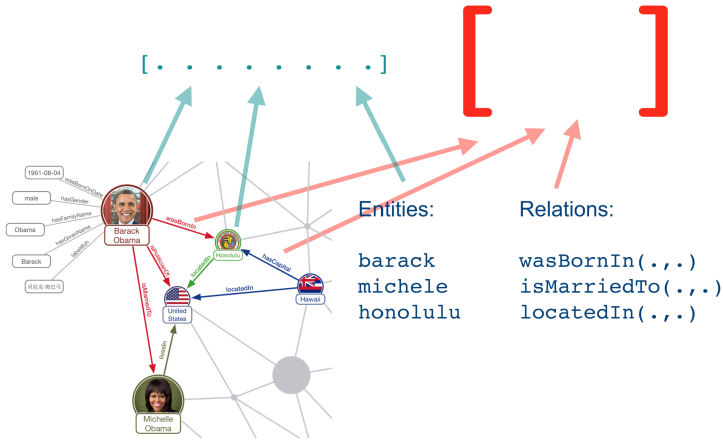


Figure 3.4: **Representing relational data with embeddings** Embedding approaches map entities (such as *barack* and *michelle*) to vectors in the Euclidean space and relations (such as *wasBornIn*) to vectors of matrices

3.3.1 Knowledge graph embeddings

By far the largest group of methods for representation learning with relational data are *knowledge graph embeddings* (KGE). Knowledge graph embeddings, at their core, approximate the symbolic data (such as the one described in Chapter 2) with vectors and/or matrices in the Euclidean space. For instance, Figure 3.4 illustrates a small part of a large knowledge graph capturing the information about Barack and Michelle Obama. KGEs replace entities, the constant symbols such as *barack* and *michelle*, with n-dimensional vectors and relations, the predicate symbols such as *wasBornIn* and *isMarriedTo*, with either vectors or matrices.

The representation learning task of KGEs is the one of learning the numerical representation of symbols (Figure 3.4). To evaluate the *fitness* of a numerical representation, KGE associate a *score* with each fact of the form *predicate(subject,object)*. The vast variety of KGE methods comes from simply modifying the score function. The existing score functions largely belong to two groups of approaches:

- **Translational approaches:** these approaches interpret relations as *translations* between objects. That is, the score measure how well the vector representation of the *head* object translates to the vector representation of the *tail* object through the given *relation*: *head* + *relation*

$\approx$ *tail*:

$$s(h, r, t) = -||\mathbf{e}_h + \mathbf{e}_r - \mathbf{e}_t|| \tag{3.4}$$

Examples of such approaches are *transE* [14] and its variants [86, 101, 2, 66]

- **Latent feature matching:** these approaches measure how well the *latent features* of objects, i.e. the individual components in vector representations, match. This is typically done by a pairwise comparison of feature values:

$$s(h, r, t) = \mathbf{e}_h \cdot \mathbf{R} \cdot \mathbf{e}_t^{\mathrm{T}} \tag{3.5}$$

A typical representatives of this KGE branch are *RESCAL* [103], *DistMult* [159] and *complEX* [140].

The main advantage of KGEs is that, in principle complex, logical reasoning is replaced with a simple algebraic computation. This makes KGEs a very scalable approach for relational learning. The main limitation of these methods is that they only approximate the original structured data, and it is difficult to quantify the *quality* of the approximation. Moreover, as the learning problem is defined in terms of *scores* of the facts, KGEs cannot easily incorporate long-range dependencies between objects. In contrast to SRL models such as Problog and Markov logic networks which can perform many types of inferences, KGEs can only be used for ranking the possible answer for a knowledge base completion tasks. Relying on score for evaluating the truthfulness of statements is related to the idea of *kernel functions* that estimate the similarity of structured objects [138, 137].

3.3.2 Capturing data properties with random walks

KGEs aim at completely replacing the relational or graph structure information with vectors. An alternative approach is to use parts of the relational information directly as features.

One such approach is *DeepWalk* [109], which combines the ideas of random walks in a graph with language modelling methodologies. In order to learn the vector representations of objects in a graph, *DeepWalk* first performs random walks collecting the sequences of nodes. It then builds the probabilistic model of node traversals by means of *SkipGram* [91], a popular word embeddings method based on the co-occurrences of words.

As *DeepWalk* uses sequences of nodes of a graph, it is inherently *transductive* and the method cannot be directly applied in inductive settings. To circumvent the

issues, methods such as *Path-ranking Algorithm* [83, 84], *Discriminative Gaifman models* [104] and *Relational Restricted Boltzmann machines* [70] rely on random walks like *DeepWalk* but use sequences of the traversed edge labels instead of nodes. Therefore, these methods can be easily applied in the inductive settings.

Path-ranking Algorithm is a model oriented towards knowledge base completion tasks and it considers the task of ranking target nodes y with respect to a query node x. It starts by enumerating a large collection of length-bounded paths, followed by estimation of the conditional probability distribution for each rule. The discovered rules are considered to be the *ranking experts*, which are combined by means of logistic regression.

Discriminative Gaifman models introduces a novelty of learning *ego-centric* representations of relational data by first extracting small bounded neighbourhoods (so-called the *Gaifman graph* [51]) around each object in the data followed by extraction of relational features from the bounded neighbourhoods. The discovered relational features are associated with the elements in a vector and are encoding the existence of the features or the count of the relational features within the bounded neighbourhood. This offers a trade-off between efficiency (how *fast* we can do the learning) and expressiveness (how much we can *express* with the given language).

Relational Restricted Boltzmann machines follow the idea very similar to the Gaifman models. The main difference is that they do not consider the relational features only within the Gaifman graph but within the entire data. Moreover, Relational Boltzmann machines use pathfinding as a relational features extraction method, while the Gaifman models can in principle use any ILP learning method.

3.3.3 Logic-based approaches

The previously discussed embeddings approaches are by far the predominant paradigm when learning latent representations of relational data. Developing logic-based methods for relational representation learning attracted significantly less attention, but offers many benefits as discussed in Chapter 1. Two examples are *Lifted relational neural networks* (LRNN) [133] and RelNN [71].

LRNNs provide a template language that maps fuzzy logical rules and data to big neural networks, in very much the same way how Markov logic networks provide a language to map complex SRL models to Markov networks. The structure of the model, i.e., the logical rules, therefore define the structure of a neural network (Figure 3.5, left). The mapping between logical and neural constructs is dictated via different types of neurons:

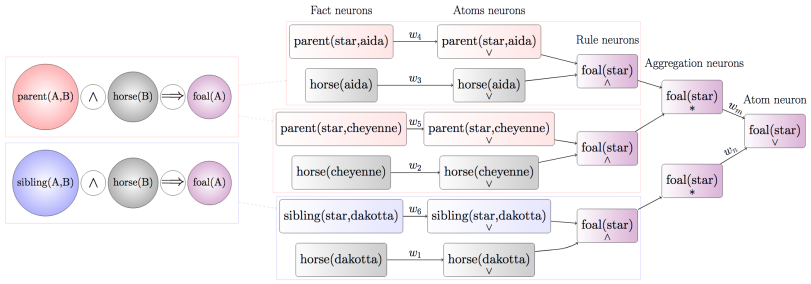


Figure 3.5: Lifted relational neural networks provide a *template* model (left) for constructing a grounded relational neural network (right) [133]

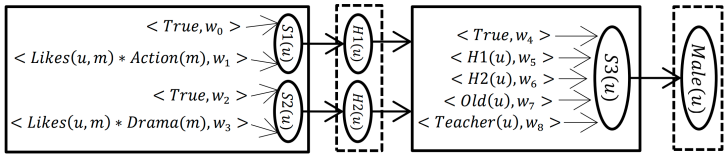


Figure 3.6: **Relational Neural Networks** perform computation by composing layers of relational logistic regression models [71]

- each ground fact is associated with a *fact neuron* taking no inputs and with value w_k (corresponding to a weight in a ANN) as output
- each ground atom is associated with an *atom neuron*
- each ground rule is associated with a *rule neuron*
- different instantiations of the same rule are handled through *aggregation neurons*.

The final structure of the LRNN is dictated by the data (Figure 3.5, right). Additionally, parameter learning is inherited from the neural networks, as well as inference.

RelNN takes a different approach and constructs relational neural networks out of relational logistic regression models as a basic primitive (Figure 3.6). This is very similar to the functioning of ANNs in which an activation of each hidden neuron can be seen as a prediction made through the logistic regression, but with a difference that the features are relational. For instance, $S1(u)$, $S2(u)$ and $S3(u)$ predicates in Figure 3.6 are relational logistic regression models jointly defining the relational neural network.

Predicate invention

The ILP community has long recognised the importance of representation change under the problem known as *predicate invention* [81]. It concerns the problem of extending the initial vocabulary that is given to a relational learner by discovering novel concepts and relations from data, which should be explained in terms of the observable, given predicates. The invented predicates can also be *statistical* if the uncertainty in the discovered predicates is represented explicitly.

The existing body of work explores several different ideas, but with limited progress. The ideas within the ILP community focus on inventing predicates through syntactic manipulation. One such idea invents predicate by analysing clauses discovered so far and forms a predicate to represent either their commonalities [158] or their differences [95]. A weakness of such approaches is that they are prone to over-generating predicates, many not useful ones. Predicates can also be invented by instantiating second-order templates [132], or to represent exceptions to learned rules [134].

One of the most successful approaches to predicate invention is recently introduced *meta-interpretative* learning framework (MIL) [98, 23, 21]. It is based on Prolog's meta-interpreter, Prolog's ability to access its own code. The core idea is very simple: the user specifies a set of *templates* imposing the structure on the potential clauses and the meta-interpreter *fills in* the templates with the predicates in the dataset, generating a logical theory. MIL has proven to be a very effective approach for relational representation change in many domains, from learning robotic strategies to string transformations [22, 96, 24]

Within the SRL community, Popescul and Unger [112] apply k-means clustering to the objects of each type in a domain, create predicates to represent clusters, and learn relations among them. Perlich and Provost [107] present a number of approaches for aggregating multi-relational data. Craven and Slattery [20] propose a learning mechanism for hypertext domains in which class predictions produced by naive Bayes are added to an ILP system (FOIL) as invented predicates. Davis et al. [26] learn Horn clauses with an off-the-shelf ILP system, create a predicate for each clause learned, and add it as a feature to the database. Kok and Domingos [76] cluster both predicate and constant symbols to create new predicates. Wang et al. [154] capture differences between similar formulas and represent it with a new predicate.

3.3.4 Hybrid approaches

Previously discussed approaches proposed distinct ideas for relational representation learning, each with its strengths and weaknesses. This has inspired research combining the ideas from the above-outlined directions.

Most of the research in this direction focuses on imposing certain logical properties into embedded spaces. Such properties include inversion and equivalence axioms [92] and logical rules [30, 122, 120]. Another line of research [93] uses the adversarial learning principle in order to impose the regularisation on the embedding spaces. This approach requires a domain knowledge that specifies Horn clauses that are known to hold for the specific domain, and it introduces an *inconsistency loss* that measures to which extent the clauses are satisfied by the embedding model.

An interesting approach towards combining logical and embedding approaches for relational representation learning comes from *differentiable theorem proving* [121]. This approach starts from an observation that the standard theorem proving procedures (e.g., Prolog) are often too brittle for usage with real-world knowledge bases which are not perfect and contain lots of noise. As standard theorem proving techniques operate purely on the syntactic level, such techniques cannot detect semantic similarity between predicates such as *grandfatherOf* and *grandpa*. This is the scenario that differentiable theorem proving addresses: *how can we enable such semantic interpretation of the symbols in the knowledge base?*

In order to enable that, it introduces *soft unification*. Unification is the core concept logic programming with the task of checking whether two terms can be represented with the same structure. For instance, we would like the term `diet(X,Y)` to unify with a fact from the knowledge base, e.g., `diet(cat,carnivore)`. A simplified version of unification between two terms $\mathtt{T}_1$ and $\mathtt{T}_2$ proceeds as follows:

1. if $\mathtt{T}_1$ and $\mathtt{T}_2$ are both constants: if they are the same then unification succeeds, otherwise fail
2. if $\mathtt{T}_1$ is a variable, instantiate $\mathtt{T}_1$ to $\mathtt{T}_2$
3. if $\mathtt{T}_2$ is a variable, instantiate $\mathtt{T}_2$ to $\mathtt{T}_1$
4. if $\mathtt{T}_1$ and $\mathtt{T}_2$ are complex terms of the same arity and predicate symbol, take the ordered set of arguments of both $\mathtt{T}_1$ and $\mathtt{T}_2$. For each pair of arguments $\mathtt{A}_i$ and $\mathtt{B}_i$ from the same position in the term, $\mathtt{A}_i$ must unify with $\mathtt{B}_i$.

5. otherwise `fail`

Soft unification modifies the steps 1 and 4 and allows symbols to be matched even if they are not identical (*grandfatherOf* and *grandpa*) but similar in meaning. This soft match then reduces the confidence of the unification proportionally to the distance between embeddings of the matched symbols in the Euclidean space.

Chapter 4

An expressive dissimilarity measure for relational clustering

This chapter introduces a novel framework for relational data clustering. The main contribution of this chapter is a new dissimilarity measure for relational objects based on the concept of a neighbourhood tree – a compact summary of the neighbourhood of a relational object. The proposed relational clustering framework will serve as a basis for the relational latent representation method introduced in the next chapter. This chapter is based on the following publications:

> DUMANČIĆ, S., AND BLOCKEEL, H. An efficient and expressive similarity measure for relational clustering using neighbourhood trees. In *ECAI*, vol. 285 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 1674–1675
>
> DUMANČIĆ, S., AND BLOCKEEL, H. An expressive dissimilarity measure for relational clustering using neighbourhood trees. *Machine Learning 106*, 9-10 (Oct. 2017), 1523–1545

4.1 Introduction

Clustering is an under-specified learning task: there is no universal criterion for what makes a good clustering, it is inherently subjective. This is known for i.i.d. data [45], and is even more true for relational data where the data set contains instances with relationships between them. Different methods for relational clustering have very different biases, which are often left implicit; for instance, some methods represent the relational information as a graph (which means they assume a single binary relation) and assume that similarity refers to proximity in the graph, whereas other methods that take the relational database stance assume the similarity comes from the relationships objects participate in. Such strong implicit biases make use of a clustering algorithm difficult for a problem at hand, without a deep understanding of both the clustering method and the problem at hand.

In this chapter, we propose a very versatile framework for clustering relational data that makes the underlying biases transparent to a user. It views a relational data set as a graph with typed vertices, typed edges, and attributes associated to the vertices. This view is very similar to the viewpoint of relational databases or predicate logic. The task we consider is clustering the vertices of one particular type. What distinguishes our approach from other approaches is that the concept of (dis)similarity used here is very broad. It can take into account attribute dissimilarity, dissimilarity of the relations an object participates in (including roles and multiplicity), dissimilarity of the neighbourhoods (in terms of attributes, relationships, or vertex identity), and inter-connectivity or graph proximity of the objects being compared.

Consider for example Figure 4.1. This relational dataset describes people and organisations, and relationships between them (friendship, a persons's role in the organisation, ...). Persons and organisations are vertices in the graph shown there (shown as white/grey ellipses), the relationships between them are shown as edges, and their attributes are shown in dashed boxes. Vertices can be clustered in very different ways:

1. `Google` and `Microsoft` are similar because of their attributes, and could be clustered together for that reason
2. `John`, `Rose` and `Han` form a densely interconnected cluster
3. `Bob`, `Joe` and `Rose` share the property that they fulfil the role of supervisor

Non-relational clustering systems will yield clusters such as the first one; they only look at the attributes of individuals. Graph partitioning systems yield

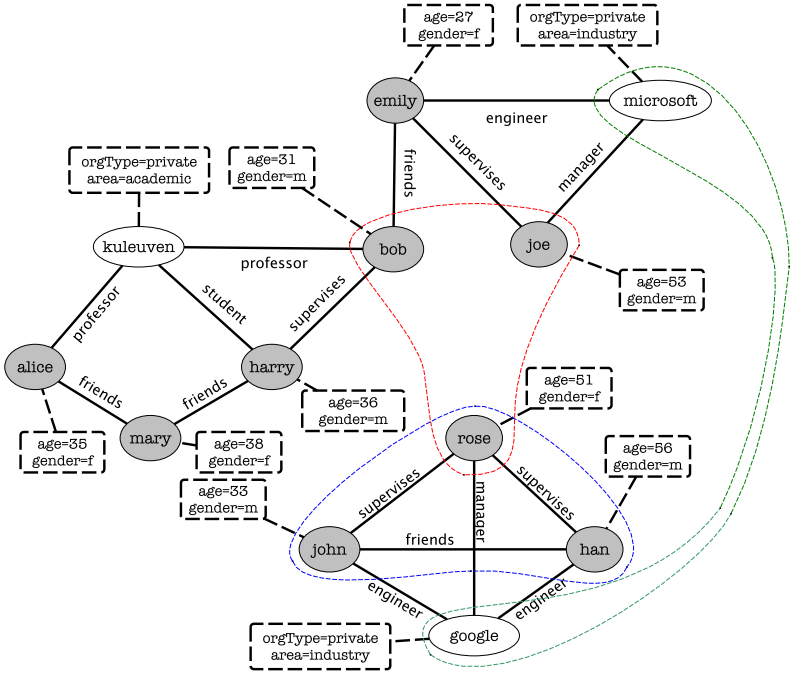


Figure 4.1: An illustration of a relational dataset containing people and organisations, and different clusters one might find in it. Instances - people (gray vertices) and organisations (white vertices), are represented by vertices, while relationships among them are represented with edges. The rectangles list an associated set of attributes for the corresponding vertex.

clusters of the second type. Some relational clustering systems yield clusters of the third type, which are defined by local structural properties. Most existing clustering systems have a very strong bias towards "their" type of clusters; a graph partitioning system, for instance, cannot possibly come up with the {Google, Microsoft} cluster, since this is not a connected component in the graph. The new clustering approach we propose is able to find all types of clusters, and even clusters that can only be found by mixing the biases.

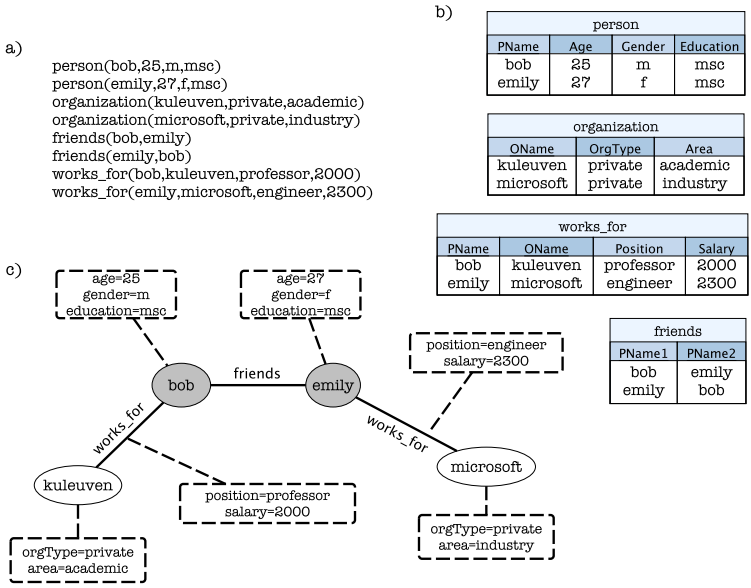


Figure 4.2: **Representation paradigms of relational data.** *Logical view* (Section *a)*) represents the relational data set as a set of logical facts; the upper part represents the definition of each predicate, while the bottom part lists all facts. *Database view* (section *b)*) associates a single database table with each of the predicates in the logical view. In the *graph view* (section *c)*) each node (circle) represents an instance and an associated set of attributes (represented with a rectangle), while relations are represented by the edges.

4.2 Relational clustering over neighbourhood trees

4.2.1 Hyper-graph Representation

Within relational learning, at least three different paradigms exist: inductive logic programming [97], which uses first-order logic representations; relational data mining [43], which is set in the context of relational databases; and graph mining [19], where relational data are represented as graphs. We illustrate the different types of representation in Figure 4.2. This example represents a set of people and organisations, and the relationships between them. The relational database format *(b)* is perhaps the most familiar to most people. It has

a table for each entity type (`Person, Organisation`) and for each relationship type between entities (`Works_for, Friends`). Each table contains multiple attributes, each of which can be an identifier for a particular entity (a *key attribute*, e.g., `PName`), or a property of that entity (`Age,Gender,...`). The logic-based format *(a)* is very similar; it consists of logical facts, where the predicate name corresponds to the table's name and the arguments to the attribute values. There is a one-to-one mapping between rows in a table and logical facts. The logic based view allows for easy integration of background knowledge (in the form of first-order logic rules) with the data. Finally, there is the attributed graph representation *(c)*, where entities are nodes in the graph, binary relationships between them are edges, and nodes and edges can have attributes. This representation has the advantage that it makes the entities and their connectivity more explicit, and it naturally separates identifiers from real attributes (e.g., the `PName` attribute from the `Person` table is not listed as an attribute of `Person` nodes, because it only serves to uniquely identify a person, and in the graph representation the node itself performs that function). A disadvantage is that edges in a graph can represent only binary relationships.

Though the different representations are largely equivalent, they provide different views on the data, which affects the clustering methods used. For instance, a notion such as shortest path distance is much more natural in the graph view than in the logic-based view, while the fact that there are different types of entities is more explicit in the database view (one table per type). The distinction between entities and attribute values is explicit in the graph, but more implicit in the database view (key vs. non-key attributes) and absent in the logic view.

In this chapter, we will use a hyper-graph view that combines elements of all the above. An oriented hyper-graph is a structure $H = (V, E)$ where V is a set of vertices and E a set of hyper-edges; a hyper-edge is an ordered multi-set whose elements are in V. Directed graphs are a special case of oriented hyper-graphs where all hyper-edges have cardinality two.

A set of relational data is represented by a typed, labelled, oriented hyper-graph (V, E, τ, λ) with V a set of vertices, E a set of hyper-edges, and $\tau : (V \cup E) \to T_V \cup T_E$ a type function that assigns a type to each vertex and hyper-edge (T_V is the set of vertex types, T_E the set of hyper-edge types). With each type $t \in T_V$ a set of attributes $A(t)$ is associated, and λ maps each vertex v to a vector of values, one value for each attribute in $A(\tau(v))$. If $a \in A(\tau(v))$, we write $a(v)$ for the value of a in v.

A relational database can be converted into the hyper-graph representation as follows.[1] For each table with only one key attribute (describing the entities

[1] For the logic-based representation, the conversion is analogous.

identified by that key), a vertex type is introduced, whose attributes are the non-key attributes of the table. Each row becomes one vertex, whose identifier is the key value and whose attribute values are the non-key attribute values in the row. For each table with more than one key attribute (describing non-unary relationships among entities), a hyper-edge is introduced that contains the vertices corresponding to these entities in the order they occur in the table. Our hyper-graph representation does not associate attributes with hyper-edges, only with vertices; hence, for non-unary relationships contain non-key attributes, a new vertex type corresponding to that hyper-edge type is introduced.

The clustering task we consider is the following: given a vertex type $t \in T_V$, partition the vertices of this type into clusters such that vertices in the same cluster tend to be similar, and vertices in different clusters dissimilar, for some subjective notion of similarity. In practice, it is of course not possible to use a subjective notion; one uses a well-defined similarity function, which hopefully on average approximates well the subjective notion that the user has in mind. The following section introduces *neighbourhood trees*, a structure we use to compactly represent and describe a neighbourhood of a vertex.

4.2.2 Neighbourhood tree

A neighbourhood tree is a tree rooted at a vertex of interest, i.e. the vertex whose neighbourhood one wants to describe. It is constructed simply by following the hyper-edges from the root vertex, as outlined in Algorithm 2. The construction of the neighbourhood tree is parametrised with the pre-specified depth, a vertex of interest and the original hyper-graph. Consider a vertex v. For every hyper-edge E in which v participates (lines 7-13), add a directed edge from v to each vertex $v' \in E$ (line 9). Label each vertex with its type, and attach to it the corresponding attribute vector (line 10). Label the edge with the hyper-edge type and the position of v in the hyper-edge (recall that hyper-edges are ordered sets; line 11). The vertices thus added are said to be at depth 1. If there are multiple hyper-edges connecting vertices v and v', v' is added each time it is encountered. Repeat this procedure for each v' on depth 1 (stored in variable `toVisit`). The vertices thus added are at depth 2. Continue this procedure up to some predefined depth d. The root element is never added to the subsequent levels, while other vertices are added as encountered. An example of a neighbourhood tree is given in Figure 4.3.

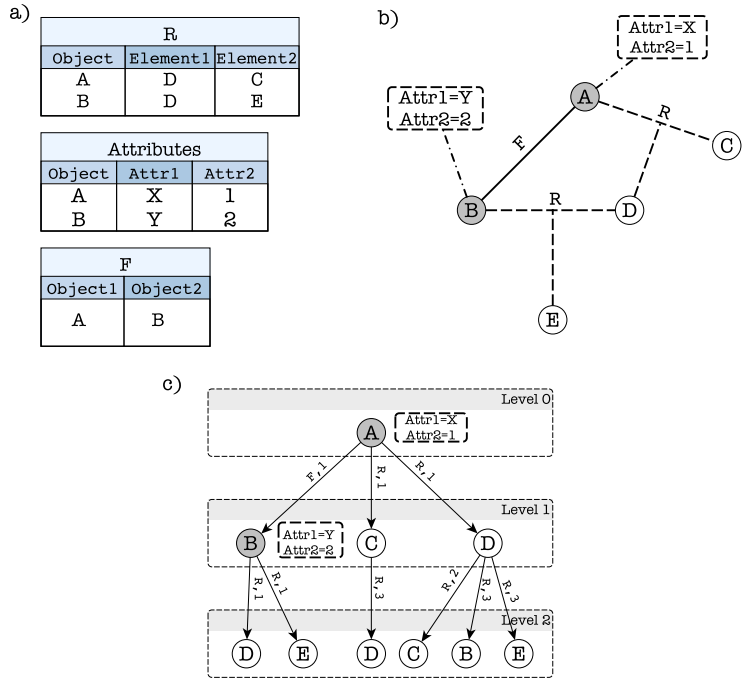


Figure 4.3: **An illustration of the neighbourhood tree.** The domain contains two types of vertices - *objects* (`A` and `B`) and *elements* (`C`, `D` and `E`), and two fictitious relations: `R` and `F`. The vertices of type *object* have an associated set of attributes. Section *a)* contains the database view of the domain. Section *b)* contains the corresponding hyper-graph view. Here, edges are represented with full lines, while hyper-edges are represented with dashed lines. Finally, section *c)* contains the corresponding *neighbourhood tree* for the vertex `A`.

Algorithm 2: Neighbourhood tree construction

Data: a hyper-graph $H = (V, E, \tau, \lambda)$
a vertex of interest v
a depth d
Result: a neighbourhood tree NT

```
/* initialize neighbourhood tree                                    */
NT = new neighbourhood tree;
NT.addRoot(v);
NT.labelVertex(v);                  /* add type and attributes */
toVisit = {v};                          /* vertices to process */
d' = 1;                                    /* depth indicator */
/* repeat until the pre-specified depth                             */
while d' ≤ d do
    foreach v' ∈ toVisit do
        foreach outgoing edge e of vertex v' do
            foreach vertex v'' in hyper-edge e do
                NT.addVertex(v'');
                NT.addEdge(v',v'');
                NT.labelVertex(v'');    /* add type and attributes */
                NT.labelEdge(v', v'');  /* add edge type and position
                 */
                toVisit = toVisit ∪ {v''};
            end
        end
        toVisit = toVisit \ {v',v}
    end
    d' += 1;
end
```

4.3 Dissimilarity measure

The main idea behind the proposed dissimilarity measure is to express a wide range of similarity biases that can emerge in relational data, as discussed and exemplified in Section 4.1. The proposed dissimilarity measure compares two vertices by comparing their neighbourhood trees. It does this by comparing, for each level of the tree, the distribution of vertices, attribute values, and outgoing edge labels observed on that level. Earlier work in relational learning has shown that distributions are a good way of summarising neighbourhoods [108].

The method for comparing distributions distinguishes between discrete and continuous domains. For discrete domains (vertices, edge types, and discrete attributes), the distribution simply maps each value to its relative frequency in the observed multi-set of values, and the χ^2-measure for comparing distributions [160] is used. That is, given two multi-sets A and B, their dissimilarity is defined as

$$d(A,B) = \sum_{x \in A \cup B} \frac{(f_A(x) - f_B(x))^2}{f_A(x) + f_B(x)} \tag{4.1}$$

where $f_S(x)$ is the relative frequency of element x in multi-set S (e.g., for $A = \{a, b, b, c\}$, $f_A(a) = 0.25$ and $f_A(b) = 0.5$).

For attributes with continuous domains, we compare distributions by applying aggregate functions to the multi-set of values, and comparing these aggregates. Given a set $\mathcal{A}$ of aggregate functions, the dissimilarity is defined as

$$d(A,B) = \sum_{f \in \mathcal{A}} \frac{f(A) - f(B)}{r} \tag{4.2}$$

with r a normalisation constant ($r = \max_M f(M) - \min_M f(M)$, with M ranging over all multi-sets for this attribute observed in the entire set of neighbourhood trees). In our implementation, we use the mean and standard deviation as aggregate functions.

The above methods for comparing distributions have been chosen for their simplicity and ease of implementation. More sophisticated methods could be used. The main point of this section, however, is *which* distributions are compared, not *how* they are compared.

We use the following notation. For any neighbourhood tree g,

- $V^l(g)$ is the multi-set of vertices at depth l (the root having depth 0)
- $V^l_t(g)$ is the multi-set of vertices of type t at depth l
- $B^l_{t,a}(g)$ is the multi-set of values of attribute a observed among the nodes of type t at depth l
- $E^l(g)$ is the multi-set of edge types between depth l and $l+1$

E.g., for the neighbourhood tree in Figure 4.3, we have

- $V^1(g) =$ {`B, C, D`}

- $V^1_{object}(g) = \{\texttt{B}\}$
- $E^1(g) = \{(\texttt{F},\texttt{1}), (\texttt{R},\texttt{1}), (\texttt{R},\texttt{1})\}$
- $B^1_{object,Attr1}(g) = \{\texttt{Y}\}$

Let $\mathcal{N}$ be the set of all neighbourhood trees corresponding to the vertices of interest in a hyper-graph. Let $norm(\cdot)$ be a *normalisation operator*, defined as

$$norm(f(g_1, g_2)) = \frac{f(g_1, g_2)}{\max\limits_{g,g' \in \mathcal{N}} f(g, g')},$$

i.e., the normalisation operator divides the value of the function $f(g_1, g_2)$ of two neighbourhood trees g_1 and g_2 by the highest value of the function f obtained amongst all pairs of neighbourhood trees.

Intuitively, the proposed method starts by comparing two vertices according to their attributes. It then proceeds by comparing the properties of their neighbourhoods: which vertices are in there, which attributes they have and how are they interacting. Finally, it looks at the proximity of vertices in a given hyper-graph. Formally, the dissimilarity of two vertices v and v' is defined as the dissimilarity of their neighbourhood trees g and g', which is:

$$\begin{aligned} s(g, g') = & w_1 \cdot \mathsf{ad}(g, g') + w_2 \cdot \mathsf{nad}(g, g') + w_3 \cdot \mathsf{cd}(g, g') \\ & + w_4 \cdot \mathsf{nd}(g, g') + w_5 \cdot \mathsf{ed}(g, g') \end{aligned} \tag{4.3}$$

where $\sum_i w_i = 1$ and

- *attribute-wise dissimilarity* (Figure 4.4, red)

$$\mathsf{ad}(g, g') = norm\left(\sum_{a \in A(\tau(v))} d(B^0_{t,a}(g), B^0_{t,a}(g'))\right) \tag{4.4}$$

  measures the dissimilarity of the root elements v and v' according to their attribute-value pairs.

- *neighbourhood attribute dissimilarity* (Figure 4.4, blue)

$$\mathsf{nad}(g, g') = norm\left(\sum_{l=1}^{d} \sum_{t \in T_V} \sum_{a \in A(t)} d(B^l_{t,a}(g), B^l_{t,a}(g'))\right) \tag{4.5}$$

measures the dissimilarity of attribute-value pairs associated with the neighbouring vertices of the root elements, per level and vertex type.

- *connection dissimilarity*

$$\mathsf{cd}(g, g') = 1 - norm\left(|\{v \in V^0(g) | v \in V^1(g')\}|\right) \tag{4.6}$$

reflects the number of edges of different type that exist between the two root elements.

- *neighbourhood dissimilarity* (Figure 4.4, green)

$$\mathsf{nd}(g, g') = norm\left(\sum_{l=1}^{\#levels} \sum_{t \in T_v} d(V_t^l(g), V_t^l(g'))\right) \tag{4.7}$$

measures the dissimilarity of two root elements according to the vertex identities in their neighbourhoods, per level and vertex type.

- *edge distribution dissimilarity* (Figure 4.4, purple):

$$\mathsf{ed}(g, g') = norm\left(\sum_{l=1}^{\#levels} d(E^l(g), E^l(g'))\right) \tag{4.8}$$

measures the dissimilarity over edge types present in the neighbourhood trees, per level.

Each component is normalised to the scale of 0-1 by the highest value obtained amongst all pair of vertices, ensuring that the influence of each factor is proportional to its weight. The weights w_{1-5} in Equation 4.3 allow one to formulate a bias through the similarity measure. For the remainder of the text, we will term our approach as ReCeNT (for Relational Clustering using Neighbourhood Trees). The benefits and downsides of this formulation are discussed and contrasted to the existing approaches in Sections 4.4.3 and 4.5.3.

This formulation is somewhat similar to the *multi-view clustering* [10], with each of the components forming a different view on data. However, there is one important fundamental difference: multi-view clustering methods want to find clusters that are good in each view separately, whereas our components do not represent different views on the data, but different potential biases, which jointly contribute to the similarity measure.

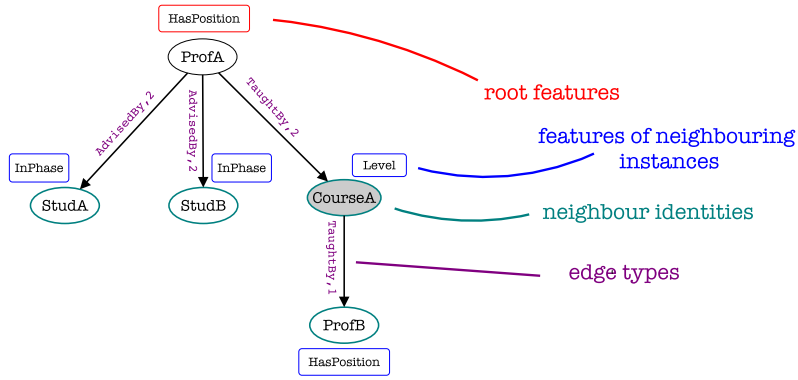


Figure 4.4: Semantic decomposition of neighbourhood tree

4.4 Positioning in the literature

4.4.1 Hyper-graph representation

Two interpretations of the hyper-graph view of relational data exist in literature. The one we incorporate here, where domain objects form vertices in a hyper-graph with associated attributes, and their relationships form hyper-edges, was first introduced by [117]. An alternative view, where logical facts form vertices, is presented by [105]. Such representations were later utilised to learn the formulas of relational models by *relational path-finding* [79, 117, 105, 87].

The neighbourhood tree introduced in Section 4.2.2 can be seen as summary of all paths in a hyper-graph originating at a certain vertex. Though neighbourhood trees and relational path-finding rely on a hyper-graph view, the tasks they solve are conceptually different. Whereas the goal of the neighbourhood tree is to compactly represent a neighbourhood of a vertex by summarising all the paths originating at the vertex, the goal of relational path-finding is to identify a small set of important paths that appear often in a hyper-graph. Additionally, a practical difference is the distinction between hyper-edges and attributes - a neighbourhood tree is constructed by following only the hyper-edges, while the mentioned work either treats attributes as unary hyper-edges or requires a declarative bias from the user.

4.4.2 Related tasks

Two problems related to the one we consider here are graph and tree partitioning [3]. Graph partitioning focuses on partitioning the original graph into a set of smaller graphs such that certain properties are satisfied. Though such partitions can be seen as clusters of vertices, the clusters are limited to vertices that are connected to each other. Thus, the problem we consider here is strictly more general, and does not put any restriction of that kind on the cluster memberships; the (dis)similarity of vertices can originate in any of the (dis)similarity sources we consider, most of which cannot be expressed within a graph partitioning problem.

A number of tree comparison techniques [11] exists in the literature. These approaches consider only the identity of vertices as a source of similarity, while ignoring the attributes and types of both vertices and hyper-edges. Thus, they are not well suited for the comparison of neighbourhood trees.

4.4.3 Relational clustering

The relational learning community, as well as the graph kernel community have previously shown interest in clustering relational (or structured) data. Existing similarity measures within the relational learning community can be coarsely divided into two groups.

The first group consists of similarity measures defined over an attributed graph model [110], with examples in *Hybrid similarity (HS)* [99] and *Hybrid similarity on Annotated Graphs (HSAG)* [157]. Both approaches focus on attribute-based similarity of vertices where HS compares the attributes of all connected vertices, and HSAG's similarity measure compares attributes of the vertices themselves and attributes of their neighbouring vertices. The main limitations of these approaches are that they ignore the existence of vertex and edge types, and impose a very strict bias towards attributes of vertices. In comparison to the presented approach, HS defines dissimilarity as the ad component if there is an edge between two vertices, and ∞ otherwise. HSAG defines the dissimilarity as a linear combination of the ad and nad components for each pair of vertices.

In contrast to the first group which employs a graph view, the second group of methods employs a predicate logic view, The two most prominent approaches are *Conceptual clustering of Multi-relational Data* (CC) [47] and *Relational instance-based learning* (RIBL) ([44], [75]). CC firstly describes each example (corresponding to a vertex in our problem) with a set of logical clauses that can be generated by a *bottom clause saturation* [16]. The obtained clauses

are considered as features, and their similarity is measured by the *Tanimoto similarity* - a measure of overlap between sets. In that sense, it is similar to using the ad and ed components for generating clauses. Note that this approach does not differentiate between relations (or interactions) and attributes, does not consider distributions of any kind, and does not have a sense of depth of a neighbourhood. Finally, RIBL follows an intuition that the similarity of two objects depends on the similarity of their attributes' values and the similarity of the objects related to them. To that extent, it first constructs a *context descriptor* - a set of objects related to the object of interest, similarly to the neighbourhood trees. Comparing two object now involves comparing their features and computing the similarity of the set of objects they are linked to. That requires matching each object of one set to the most similar object in the other set, which is an expensive operation (proportional to the product of the set sizes). In contrast, the χ^2 distance is linear in the size of the multi-set. Further, the χ^2 distance takes the multiplicity of elements into account (it essentially compares distributions), which the RIBL approach does not.

Within the graph kernel community, two prominent groups exist: *Weisfeiler-Lehman graph kernels (WL)* [129, 128, 49, 61, 4] and *random walk based kernels* [152, 87]. A common feature of these approaches is that they rely on global structural properties of graphs. The Weisfeiler-Lehman Graph Kernels are a family of graph kernels developed upon the *Weisfeiler-Lehman isomorphism test*. The key idea of the WL isomorphism test is to extend the set of vertex attributes by the attributes of the set of neighbouring vertices, and compress the augmented attribute set into new set of attributes. There each new attribute of a vertex corresponds to a subtree rooted from the vertex, similarly to the neighbourhood trees. Shervashidze and Borgwardt have introduced a fast WL subtree kernel (WLST) [128] for un-directed graphs by performing the WL isomorphism test to update the vertex labels, followed by counting the number of matched vertex labels. WL family of graph kernels, in contrast to our approach, extends the set of attributes by identifying isomorphic sub-trees present in (sub)graphs. This is reflected in the bias they impose, that is, the similarity comes from the structure of a graph (in our case, a neighbourhood tree).

A Rooted Kernel for Ordered hyper-graph (RKOH) [152] is an instance of random walk kernels successfully applied in relational learning tasks. These approaches estimate the similarity of two (hyper)graphs by comparing the walks one can obtain by traversing the hyper-graph. RKOH defines a similarity measure that compares two hyper-graphs by comparing the paths originating at every edge of both hyper-graphs, instead of the paths originating at the root of the hyper-graph. RKOH does not differentiate between attributes and hyper-edges, but treats everything as an hyper-edge instead (an attribute can be seen as an unary edge).

Table 4.1: **Aspects of similarity considered by various relational clustering approaches.** ✓ denotes full consideration, ≃ partial and × no consideration at all.

Similarity	Attributes	Neighbour. attributes	Neighbour. identities	Proximity	Structural properties
ReCeNT	✓	✓	✓	✓	✓
HS	✓	×	×	×	×
HSAG	✓	✓	×	×	×
RIBL	✓	✓	✓	×	×
CC	≃	≃	×	×	≃
RKOH	×	≃	×	×	✓
WLST	×	≃	×	×	✓

Table 4.1 summarises different aspects of similarity considered by the above mentioned approaches. The interpretations of similarity are divided into five sources of similarity. The first two categories concern attributes: attributes of the vertices themselves and their neighbouring vertices. The following two categories concern identities of vertices in the neighbourhood of a vertex of interest. They concern sub-graphs (identity of vertices in the neighbourhood) centred at a vertex, and proximity of two vertices. The final category concerns the structural properties of sub-graphs in the neighbourhood of a vertex defined by the neighbourhood tree.

4.4.4 Complexity analysis

Though scalability is not the focus of this work, here we show that the proposed approach is as scalable as the state-of-the-art kernel approaches, and substantially less complex than the majority of the above-mentioned approaches that use both attribute and link structure. For the sake of clarity of comparison, assume a homogeneous graph with only one vertex type and one edge type. Let N be the number of vertices in a hyper-graph, L be the total number of hyper-edges, and d be the depth of a neighbourhood representation structure, where applicable. Let, as well, A be the number of attributes in a data set. Additionally, assume that all vertices participate in the same number of hyper-edges, which we will refer to as E. This might seem like a strong assumption, but suffices for the purpose of the analysis. We will refer to the length of clause in CC and path in RKOH as l.

To compare any two vertices, ReCeNT requires one to compute the dissimilarity

Table 4.2: Complexities of the various methods for relational clustering

Approach	Complexity
HS	$O(LA)$
HSAG	$O\left(N^2EA\right)$
ReCeNT	$O\left(N^2E^d\right)$
WLST	$O\left(N^2E^d\right)$
CC	$O\left(N^2\binom{E+A}{l}\right)$
RIBL	$O\left(N^2\prod_{k=1}^{d}(E+A)^{2k}\right)$
RKOH	$O\left(N^2\left(E+A\right)^{2d+2l}\right)$

of the multi-sets representing the vertices, proportional to $O(d \times A + \sum_{k=1}^{d} E^k) = O\left(N^2E^d\right)$. Table 4.2 summarises the complexities of the discussed approaches. In summary, the approaches can be grouped into three categories. The first category contains HS and HSAG; these are substantially less complex than the rest, but focus only on the attribute similarities. The second category contains RIBL and RKOH, which are substantially more complex than the rest. Both of these approaches use both attribute and edge information, but in a computationally very expensive way. The last category contains ReCeNT, WLST and CC; these lie in between. They utilise both attribute and edge information, but in a way that is much more efficient than RIBL and RKOH.

The complexity of ReCeNT benefits mostly from two design choices: *differentiation of attributes and hyper-edges*, and *decomposition of neighbourhood elements into multi-sets*. By distinguishing hyper-edges from attributes, ReCeNT focuses on identifying sparse neighbourhoods. Decomposition of neighbourhoods into multi-sets allows ReCeNT to compute the similarity linearly in the size of a multi-set. The parameter that ReCeNT is the most sensitive to is the depth of the neighbourhood tree, which is the case with the state-of-the-art kernel approaches as well. However, the main underlying assumption behind ReCeNT is that important information is contained in small local neighbourhoods, and ReCeNT is designed to utilise such information.

Table 4.3: Characteristics of the data sets used in experiments. The characteristics include the total number of vertices, the number of vertices of interest, the total number of attributes, the number of attributes associated with vertices of interest, the number of hyper-edges as well as the number of different hyper-edge types.

	Dataset				
Property	IMDB	UW-CSE	Muta	WebKB	Terror
vertices	298	734	6124	3880	1293
target vertices	268	272	230	920	1293
vertex types	3	4	2	2	1
attributes	3	7	7	1207	106
target attributes	3	3	4	763	106
hyper-edges	715	1834	30804	5779	3743
hyper-edge types	3	6	7	4	2

4.5 Evaluation

4.5.1 Data sets

We evaluate our approach on five data sets for relational clustering with different characteristics and domains. The chosen data sets are commonly used within the (statistical) relational learning community, and they expose different biases. The characterisation of data sets, summarised in Table 4.3, include the total number of vertices in a hyper-graph, the number of vertices of interest, the total number of attributes, the number of attributes associated with vertices of interest, the number of hyper-edges as well as the number of different hyper-edge types. The data sets range from having a small number of vertices, attributes and hyper-edges (UW-CSE, IMDB), to a considerably large number of vertices, attributes or hyper-edges (Mutagenesis, WebKB, TerroristAttack). All the chosen data sets are originally classification data sets, which allows us to evaluate our approach with respect to how well it extracts the classes present in the data set.

The IMDB[2] data set is a small snapshot of the Internet Movie Database. It describes a set of movies with people acting in or directing them. The goal is to differentiate people into two groups: *actors* and *directors*. The UW-CSE[3] data

[2]Available at http://alchemy.cs.washington.edu/data/imdb
[3]Available at http://alchemy.cs.washington.edu/data/uw-cse/

set describes the interactions of employees at the University of Washington and their roles, publications and the courses they teach. The task is to identify two clusters of people: *students* and *professors*. The Mutagenesis[4] data set, as described is Section 4.1, describes chemical compounds and atoms they consist of. Both compounds and atoms are described with a set of attributes describing their chemical properties. The task is to identify two clusters of compounds: *mutagenic* and *not mutagenic*. The WebKB[5] data set consists of pages and links collected from the Cornell University's webpage. Both pages and links are associated with a set of words appearing on a page or in the anchor text of a link. The pages are classified into seven groups according to their role, such as *personal*, *departmental* or *project* page. The final data set, termed Terrorists[6] [127], describes terrorist attacks each assigned one of 6 labels indicating the type of the attack. Each attack is described by a total of 106 distinct features, and two relations indicating whether two attacks were performed by the same organisation or at the same location.

4.5.2 Experimental setup

In the remainder of this section, we evaluate our approach. We focus on answering the following questions:

(Q1) *How well does ReCeNT perform on the relational clustering tasks compared to existing similarity measures?*

(Q2) *How relevant is each of the components?* We perform clustering using our similarity measure and setting the parameters as $w_i = 1, w_{j,j\neq i} = 0$.

(Q3) *To which extent can the parameters of the proposed similarity measure be learnt from data in an unsupervised manner?*

(Q4) *How well does ReCeNT perform compared to existing similarity measures in a supervised setting?*

(Q5) *How do the runtimes for ReCeNT compare to the competitors?*

In each experiment, we have used the aforementioned (dis)similarity measures in conjunction with spectral [100], and hierarchical [155] clustering algorithms. We have intentionally chosen two clustering approaches which assume different biases, to be able to see how each similarity measure is affected

[4] Available at http://www.cs.ox.ac.uk/activities/machlearn/mutagenesis.html
[5] Available at http://alchemy.cs.washington.edu/data/webkb/
[6] Available at http://linqs.umiacs.umd.edu/projects//projects/lbc/

by assumptions clustering algorithms make. We have altered the depth on neighbourhood trees between 1 and 2 wherever it was possible, and report both results.

We evaluate each approach using the following validation method: we set the number of clusters to be equal to the provided number of classes in each data set, and evaluate the obtained clustering with regards to how well it matches the known clustering given by the labels. Each obtained clustering is then evaluated using the *adjusted Rand index* (ARI) [115, 94]. The ARI measures the similarity between two clusterings, in this case between the obtained clustering and the provided labels. The ARI score ranges between -1 and 1, where a score closer to 1 corresponds to the better matching between the obtained clustering and the ground truth one, and hence better performance, while 0 is the chance level. For each data set, and each similarity measure, we report the ARI score they achieve. Additionally, we have set a timeout to 24 hours and do not report results for an approach that takes more time to compute.

To achieve a fair time comparison, we implemented all similarity measures (HS, HSAG, RIBL, CC, as well as RKOH) in `Scala` and optimised them in the same way, by caching all the intermediate results that can be re-used. We have used the clustering algorithms implemented in Python's `scikit-learn` package [106]. The hierarchy obtained by hierarchical clustering was cut when it has reached the pre-specified number of clusters. In the first experiment, the weights w_{1-5} were not tuned, and were set to 0.2. We have used mean and standard deviation as aggregates for continuous attributes.

4.5.3 Results

(Q1) Comparison to the existing methods

We compare ReCeNT to a pure attribute based approach (termed Baseline), HS [99], HSAG [157], CC [47], RIBL [44], as well as WLST [128], Linear kernel between vertex histograms (V), Linear kernel between vertex-edge histograms (VE) provided with [136], and RKOH [152]. The subscript in ReCeNT, HSAG, RIBL and kernel approaches denotes the depth of the neighbourhood tree (or other supporting structure). The subscript in CC denotes the length of the clauses. The second subscript in WLST and RKOH indicates their parameters: with WLST it is the h parameter indicating the number of iterations, whereas with RKOH it indicates the length of the walk.

The results of the first experiment are summarised in Table 4.4. The table contains ARI values obtained by the similarity measures for each data set and

Table 4.4: Performance of all approaches. For each similarity measure, the ARI achieved when the true number of clusters was used is reported (higher value is better). The results are shown for both hierarchical and spectral clustering, while the depth of the approaches is indicated by the subscript. The last column counts the number of wins per algorithm, where "win" means achieving the highest ARI on a data set.

Similarity	Spectral clustering					Hierarchical clustering					Win
	Muta	UWCSE	WebKB	Terror	IMDB	Muta	UWCSE	WebKB	Terror	IMDB	
Baseline	-0.03	0.20	0.25	0.17	0.05	-0.02	0.25	0.00	0.00	0.05	0
HS	-	0.06	0.10	-0.01	0.00	-	0.01	0.00	0.01	0.00	0
CC_2	0.01	0.82	0.04	0.01	0.10	0.00	0.10	0.00	0.01	0.10	0
CC_4	0.01	0.92	0.04	0.01	0.10	0.00	0.00	0.00	0.01	0.10	0
$ReCeNT_1$	**0.35**	**0.98**	**0.57**	**0.26**	**1.0**	**0.32**	**0.97**	**0.04**	0.00	0.62	**8**
$RIBL_1$	0.26	0.68	0.1	-	0.38	0.22	0.89	0.00	-	0.35	0
$HSAG_1$	0.06	0.0	0.05	0.24	-0.05	-0.01	0.10	0.01	0.00	0.04	0
$WLST_{1,5}$	0.02	0.33	0.33	0.07	0.66	0.00	-0.01	0.00	**0.27**	-0.01	1
$WLST_{1,10}$	0.02	0.33	0.32	0.11	0.31	0.00	-0.01	0.00	**0.27**	-0.01	1
V_1	0.03	0.19	0.00	0.00	0.00	0.00	-0.01	0.00	0.00	0.00	0
VE_1	0.03	0.36	0.00	0.00	**1.0**	0.00	0.01	0.00	0.00	**1.0**	2
$RKOH_{1,2}$	0.10	0.20	-	-	0.83	0.10	0.20	-	-	0.83	0
$RKOH_{1,4}$	-	-	-	-	-	-	-	-	-	-	0
$ReCeNT_2$	0.30	0.16	0.40	0.16	**1.0**	0.08	0.1	0.02	0.01	0.13	1
$RIBL_2$	-	0.68	-	-	0.78	-	0.0	-	-	0.63	0
$HSAG_2$	0.06	0.00	0.04	0.23	0.09	-0.01	0.1	0.00	0.00	0.04	0
$WLST_{2,5}$	0.01	0.02	0.52	0.11	0.31	0.00	0.02	0.00	**0.27**	-0.04	1
$WLST_{2,10}$	0.01	0.02	0.52	0.12	0.36	0.00	0.02	0.00	0.05	-0.04	0
V_2	0.07	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0
VE_2	0.00	0.38	**0.56**	0.00	0.53	0.00	0.01	0.00	0.00	0.00	1
$RKOH_{2,2}$	-	-	-	-	-	-	-	-	-	-	0
$RKOH_{2,4}$	-	-	-	-	-	-	-	-	-	-	0

clustering algorithm used. The last column of the table states the number of wins per approach. The number of wins is calculated by simply counting the number of cases where the approach obtained the highest ARI value, a "case" being a combination of a data set and a clustering algorithm. $ReCeNT_1$ wins 8 out of 10 times, and thus outperforms all other methods. The best results are achieved in combination with spectral clustering, with exception being the TerroristAttack data set where $WLST_{1,*}$ and $WLST_{2,5}$ combined with hierarchical clustering achieved the highest ARI of 0.27, in contrast to 0.26 obtained by $ReCeNT_1$. In all cases of the Mutagenesis and UWCSE data sets, $ReCeNT_1$ wins with a larger margin. However, it is important to note that in the remaining cases, the closest competitor is not always the same. In the case of IMDB data set in combination with spectral clustering, the closest competitor is VE_1 (together with $RKOH_{1,2}$), as well as in the case of WebKB in combination with spectral clustering. In the cases of the TerroristAttack data set combined with the spectral clustering, the closest competitors are $HSAG_1$ and $HSAG_2$, while in the case with hierarchical clustering our approach is outperformed by $WLST_{1,*}$ and $WLST_{2,5}$. These results show that the proposed similarity measure

performs better over wide range of different tasks and biases, compared to the remaining approaches. Moreover, when combined with the spectral clustering, ReCeNT_1 consistently performs well on all data sets, achieving the second-best result only on the TerroristAttack data set.

Each of the data sets exposes different bias, which influences the performance of the methods. In order to successfully identify mutagenic compounds, one has to consider both attribute and link information, including the attributes of the neighbours. Chemical compounds that have similar structure tend to have similar properties. This data set is more suitable for RIBL, ReCeNT and kernel approaches. ReCeNT_1 and RIBL_1 achieve the best results here[7], while kernels approaches surprisingly do not perform better than the chance level. The UW-CSE is a social-network-like data set where the task is to find two interacting communities with different attribute-values - students and professors. The distinction between two classes is made on a single attribute - professors have positions, while students do not, and the relation stating that professors advise students. This task is suitable for HS and HSAG. However, both approaches are substantially outperformed by ReCeNT_1 and CC_*. Similarly, the IMDB data set consists of a network of people and their roles in movies, which can be seen as a social network. Here, directors can be differentiated from actors by a single edge type - actors work under directors which is explicitly encoded in the data set. The type of interactions between entities matters the most, as it is not an attribute-rich data set, and is thus more suitable for methods that account for structural measures. Accordingly, ReCeNT, RIBL, $\text{WLST}_{1,*}$ and VE kernels achieve the best results.

The remaining data sets, WebKB and TerroristAttack, are entirely different in nature from the aforementioned ones. These data set have a substantially larger number of attributes, but those are not sufficient to identify relevant clusters supported by labels, that is, interactions contain important information. Such bias is implicitly present in HS, and partially assumed by kernel approaches. The results show that ReCeNT_1 and $\text{WLST}_{2,*}$ and VE_2 kernels achieve almost identical performance on the WebKB data set, while the remaining approaches are outperformed even by the baseline approach. On the TerroristAttack data set, $\text{WLST}_{1,*}$ kernel achieves the best performance, outperforming ReCeNT_1 and HSAG_1. Similarly to WebKB, other approaches are outperformed by the baseline approach.

The results summarised in Table 4.4 point to several conclusions. Firstly, given that the proposed approach achieves the best results in 8 out of 10 test cases, the results suggest that it is indeed versatile enough to capture relevant information,

[7] We were not able to make HS work on this data set as it assumes edges between compound vertices which are non-existing in this data set

Table 4.5: Performance of ReCeNT (in terms of ARI; higher values are better) with different parameter settings. The upper part of the table presents results with the neighbourhood trees with depth of 1, whereas the bottom part contains the results with depth set to 2. The parameters in italic indicate the best performance achieved.

Parameters	Spectral clustering					Hierarchical clustering				
	Muta	**UWCSE**	**WebKB**	**Terror**	**IMDB**	**Muta**	**UWCSE**	**WebKB**	**Terror**	**IMDB**
1,0,0,0,0	0.00	0.2	0.25	0.17	0.05	0.00	0.25	0.00	0.01	0.05
0,1,0,0,0	0.00	0.12	0.00	-0.01	0.00	0.00	0.52	0.00	0.00	0.00
0,0,1,0,0	0.00	0.10	0.10	0.00	0.13	0.00	0.05	0.00	0.00	0.14
0,0,0,1,0	0.30	-0.03	0.20	-0.01	0.17	0.30	0.02	0.00	0.00	0.17
0,0,0,0,1	0.25	0.07	0.02	0.00	1.0	0.24	0.17	0.00	-0.01	1.0
0.2,0.2,0.2,0.2,0.2	0.35	0.86	0.56	0.26	1.0	0.32	0.96	0.04	0.00	0.62
1,0,0,0,0	0.00	0.2	0.27	0.17	-0.05	0.00	0.00	0.00	0.00	0.05
0,1,0,0,0	0.00	0.16	0.00	-0.01	0.00	0.00	0.03	0.00	0.00	0.0
0,0,1,0,0	0.00	0.08	0.01	0.00	0.13	0.00	0.00	0.00	0.01	0.15
0,0,0,1,0	0.29	-0.03	0.2	-0.01	0.00	0.29	0.01	0.02	-0.01	0.00
0,0,0,0,1	0.27	0.03	0.02	0.00	1.0	0.00	-0.04	0.00	0.00	1.0
0.2,0.2,0.2,0.2,0.2	0.3	0.07	0.4	0.16	1.0	0.08	0.1	0.02	0.01	0.13

regardless of whether that comes from the attributes of vertices, their proximity, or connectedness of vertices, even without parameter tuning. Moreover, when combined with the spectral clustering, our approach consistently obtains good results on all data sets, while the competitor approaches achieve good results if the problem fits their bias. Secondly, the results show that one has to consider not only the bias of the similarity measure, but the bias of the clustering algorithm as well, which is evident on most data sets where spectral clustering achieves substantially better performance than hierarchical clustering. Finally, ReCeNT and most of the approaches tend to be sensitive to the depth parameter, which is evident in the drastic difference in performance when different depths are used. This suggests that increasing depth of a neighbourhood tree consequently introduces more noise. Interestingly, while the results suggest that with ReCeNT the depth of 1 performs the best, the performance of kernel methods tend to increase with the depth parameter. These results justify the basic assumption of this approach that important information is contained in small local neighbourhoods.

(Q2) Relevance of components

In the second experiment, we evaluate how relevant each of the five components in Equation 4.3 is. Table **??** summarises the results. There are only two cases (Mutagenesis and IMDB) where using a single component (if

it is the right one!) suffices to get results comparable to using all components (Table **??**). This confirms that clustering relational data is difficult not only because one needs to choose the right source of similarity, but also because the similarity of relational objects may come from multiple sources, and one has to take all these into account in order to discover interesting clusters.

These results may explain why ReCeNT almost consistently outperforms all other methods in the first experiment. First, ReCeNT considers different sources of relational similarity; and second, it ensures that each source has a comparable impact (by normalising the impact of each source and giving each an equal weight in the linear combination). This guarantees that if a component contains useful information, it is taken into account. If a component has no useful information, it adds some noise to the similarity measure, but the clustering process seems quite resilient to this. If *most* of the components are irrelevant, the noise can dominate the pattern. This is likely what happens in experiment 1 when depth 2 neighbourhood trees are used: too much irrelevant information is introduced at level two, dominating the signal at level one.

(Q3) Learning weights in an unsupervised manner

The first experiment shows that ReCeNT outperforms the competitor methods even without parameters being tuned. The second experiment shows that one typically has to consider multiple interpretations of similarity in order to obtain a useful clustering. A natural question to ask is whether the parameters could be learned from data in an unsupervised way. The possibility of tuning offers an additional flexibility to the user. If the knowledge about the right bias is available in advance, one can specify it through adjusting the parameters of the similarity measure, potentially achieving even better results than those presented in Table 4.4. However, tuning the weights in an automated and systematic way is a difficult task as there is no clear objective function to optimise in a purely unsupervised settings. Many clustering evaluation criteria, such as ARI, require a reference clustering which is not available during clustering itself. Other clustering quality measures do not require a reference clustering, but each of those has its own bias [142].

An approach that might help in this direction is the *Affinity Aggregation for Spectral Clustering* (AASC) [64]. This work extends spectral clustering to a multiple affinity case. The authors start from the position that similarity of objects often can be measured in multiple ways, and it is often difficult to know in advance how different similarities should be combined in order to achieve the best results. Thus, the authors introduce an approach that learns the weights that would, when clustered into the desired number of clusters, yield

Table 4.6: **Affinity aggregation results.** Performance is indicated in terms of ARI. The subscript indicates the depth of the neighbourhood tree.

Approach	IMDB	UWCSE	Mutagenesis	WebKB	Terror
ReCeNT_1	1.0	0.98	0.35	0.56	0.26
AASC_1	0.78	0.65	0.35	0.57	0.28
ReCeNT_2	1.0	0.07	0.3	0.4	0.16
AASC_2	0.67	0.23	0.3	0.4	0.23

the highest intra-cluster similarity. That is achieved by iteratively optimising: (1) the cluster assignment given the fixed weights, and (2) weights given a fixed cluster assignment. Thus, by treating each component in Equation 4.3 as a separate affinity matrix, this approach tries to learn their optimal combination.

We have tried AASC in ReCeNT, and the results are summarised in Table 4.6. These results lead to several conclusions. Firstly, in most cases AASC yields no substantial benefit or even hurts performance. This confirms that learning the appropriate bias (and the corresponding parameters) in an entirely unsupervised way is a difficult problem. The main exceptions are found for depth 2: here, a substantial improvement is found for UWCSE and TerroristAttack. This seems to indicate that the bad performance on depth 2 is indeed due to an overload of irrelevant information, and that AASC is able to weed out some of that. Still, the obtained results for depth 2 are not comparable to the ones obtained for depth 1. The cause of this is the non-selectiveness of ReCeNT: further we go from the root vertex, encountered information is less relevant for that vertex. Therefore, most of the information is irrelevant, but ReCeNT does not account for that. We conclude that tuning the weights in an unsupervised manner will require more sophisticated methods than the current state of the art.

(Q4) Performance in a supervised setting

The previous experiments point out that the proposed dissimilarity measure performs well compared to the existing approaches, but finding the appropriate weights is difficult. Though our focus is on clustering tasks, we can use our dissimilarity measure for classification tasks as well. The availability of labels offers a clear objective to optimise when learning the weights, and thus allows us to evaluate the appropriateness of ReCeNT for classification.

We have set up an experiment where we use a k nearest neighbours (kNN)

Table 4.7: Accuracy of the kNN classifier with different (dis)similarity measures and weight learning. The performance is expressed in terms of accuracy over the 10-fold cross validation.

Similarity	IMDB	UWCSE	Mutagenesis	WebKB	Terrorists
HS	88.08	76.66	0.00	12.78	27.51
CC	88.08	**99.85**	60.08	61.07	38.28
HSAG	88.08	95.88	77.40	12.82	75.62
ReCeNT	**100**	**100**	**85.54**	**100**	**85.60**
RIBL	**100**	77.22	76.37	84.11	-
WLST	93.60	44.94	76.37	47.35	45.56
VE	**100**	98.26	70.60	49.33	30.00
V	93.80	43.61	70.42	47.35	44.39
RKOH	95.07	67.26	60.78	-	-

classifier with each of the (dis)similarity measures. It consists of a 10-fold cross-validation, where within each training fold, an internal 10-fold cross-validation is used to tune the parameters of the similarity measure, and kNN with the tuned similarity measure is next used to classify the examples in the corresponding test fold.

The results of this experiment are summarised in Table 4.7. ReCeNT achieves the best performance on all data sets. On the IMDB data set, ReCeNT achieves perfect performance, as do RIBL and VE. On UWCSE, ReCeNT is 100% accurate; its closest competitor, CC, achieves 99.85%. From the classification viewpoint, these two data sets are easy: the classes are differentiable by one particular attribute or relation. On Mutagenesis and Terrorists, the difference is more pronounced: ReCeNT achieves around 85% accuracy, with its closest competitor (HSAG) achieving 76% or 77%. On WebKB, finally, ReCeNT and RIBL substantially outperform all the other approaches, with ReCeNT achieving 100% and RIBL 84.11%.

The remarkable performance of ReCeNT on WebKB is explained by inspecting the tuned weights. These reveal that ReCeNT's ability to jointly consider vertex identity, edge type distribution, and vertex attributes (in this case, words on webpages) are the reason why it performs so well. None of the other approaches take all three components into account, which is why they achieve substantially worse results.

These results clearly show that accounting for several views of similarity is beneficial for relational learning. Moreover, the availability of labelled information is clearly helpful and ReCeNT is capable of successfully adapting

Table 4.8: Runtime comparison in minutes (rounded up to the closest integer). The runtimes include the construction of supporting structures and time needed to calculate a similarity between each pair of vertices in a given hypergraph. Note that graph kernel measures (in italic) are obtained using the external software provided with [136]. - indicates that the calculation took more than 24 hours.

Approach	IMDB	UWCSE	Mutagenesis	WebKB	Terror
HS	1	1	-	1	1
CC_2	1	1	1	5	1
CC_4	1	1	1	8	8
$HSAG_1$	1	1	1	2	2
$HSAG_2$	1	1	1	5	2
$ReCeNT_1$	1	1	1	2	2
$ReCeNT_2$	1	1	3	10	5
$RIBL_1$	1	2	540	1320	-
$RIBL_2$	2	5	-	-	-
$WLST_{1,5}$	1	1	1	1	1
$WLST_{1,10}$	1	1	1	1	1
$WLST_{2,5}$	1	1	1	4	5
$WLST_{2,10}$	1	1	1	4	5
VE_1	1	1	1	1	2
$RKOH_{1,2}$	1	2	10	-	-
$RKOH_{1,4}$	-	-	-	-	-
$RKOH_{2,2}$	-	-	-	-	-
$RKOH_{2,4}$	-	-	-	-	-

its bias towards the needs of the data set.

(Q5) Runtime comparison

Table 4.8 presents a comparison of runtimes for each approach. All the experiments were run on a computer with 3.20 GHz of CPU power and 32 GB RAM. The runtimes include the construction of supporting structures (neighbourhood trees and context descriptors), calculation of similarity between all pairs of vertices, and clustering. The measured runtimes are consistent with the previously discussed complexities of the approaches. HS, HSAG, CC, ReCeNT and kernel approaches (excluding RKOH) are substantially more efficient than the remaining approaches. This is not surprising, as HS, HSAG and CC use very limited information. It is, however, interesting to see that ReCeNT and WLST, which use substantially more

information, take only slightly more time to compute, while achieving substantially better performance on most data sets. These approaches are also orders of magnitude more efficient than RIBL and RKOH, which did not complete on most data sets with depth set to 2. That is particularly the case for RKOH which did not complete in 24 hours even with the depth of 1, when the walk length was set to 4.

4.6 Conclusion

We have introduced a novel dissimilarity measure for clustering relational objects, based on a hyper-graph interpretation of a relational data set. In contrast with the previous approaches, our approach takes multiple aspects of relational similarity into account, and allows one to focus on a specific vertex type of interest, while at the same time leveraging the information contained in other vertices. We develop the dissimilarity measure to be versatile enough to capture relevant information, regardless whether it comes from attributes, proximity or connectedness in a hyper-graph. To make our approach efficient, we introduce neighbourhood trees, a structure to compactly represent the distribution of attributes and hyper-edges in the neighbourhood of a vertex.

We experimentally evaluate our approach on several data sets on both clustering and classification tasks. The experiments show that the proposed method often achieves better results than the competitor methods with regards to the quality of clustering and classification, showing that it indeed is versatile enough to adapt to each data set individually. Moreover, the proposed approach, though more expressive, is as efficient as the state-of-the-art approaches. One open challenge is to which extent the parameters of the proposed similarity measure can be learnt from data in an unsupervised (or a semi-supervised) way.

Intermezzo: Semi-supervised clustering with COBRAS

Clustering is a difficult task due to the lack of a clear criterion for evaluating clusterings. We have dealt with that by relying on the available supervision to provide the ground truth. *Semi-supervised clustering* starts from a position that, even though a fully labelled dataset is not available, the user is able to provide a limited amount of feedback that can be further used to guide the clustering procedure. The feedback usually takes the form of the following question: *do instances i and j have the same cluster label in the target clustering?* Answering the question with `yes` yields a *must-link* constraint and a *cannot-link* constraint otherwise.

In a series of works on `COBRAS`, Van Craenendonck et al. [143, 144, 145] introduce a fast, iterative and active clustering method with pairwise constraints. `COBRAS` is based on the concept of *super-instances*: a local set of instances that are assumed to belong to the same clustering in the final target clustering. The core idea of `COBRAS` is to ask the pairwise queries among the representatives of super-instances instead of the individual data points. Thus, it is related to the idea of *lifted inference* [111].

`COBRAS` introduces an incremental procedure that, starting with all data points in a single super-instance, repeatedly intertwines the following steps:

- select a super-instance
- repeatedly split the selected super-instance (and its descendants) in two, asking a *pairwise query* between the descendants. Stop when the first *must-link* constraint is obtained
- Ask the *pairwise query* among all available super-instances, starting with the closest ones.

This has proven to be a fast, versatile and well-performing semi-supervised clustering procedure that outperforms many of the existing semi-supervised clustering approaches on a wide range of benchmark tasks, and *merge* super-instances connected by the *must-link* constraint.

Chapter 5

Learning latent features by capturing approximate symmetries

This chapter explores the idea of learning relational representations by identifying symmetries in data – groups of relational objects *similar* to each other, and using the obtained group membership information as the latent representation of data. The proposed approach builds upon the previously introduced relational clustering method and extends it towards the relation clustering. This chapter is based on the following publications:

> DUMANČIĆ, S., AND BLOCKEEL, H. Unsupervised relational representation learning via clustering: Preliminary results. In *6th workshop on Statistical Relational Artificial Intelligence StarAI at IJCAI* (2016)
>
> DUMANČIĆ, S., AND BLOCKEEL, H. Clustering-based relational unsupervised representation learning with an explicit distributed representation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17* (2017), pp. 1631–1637
>
> DUMANČIĆ, S., AND BLOCKEEL, H. Demystifying relational latent representations. In *Inductive Logic Programming* (2018), N. Lachiche and C. Vrain, Eds., Springer International Publishing, pp. 63–77. Best Student Paper Award

5.1 Introduction

Precisely defining the properties of good data representation is difficult, if not impossible. We have seen some of the intuitive notions in Section 3.2.3, but many of them are difficult to define precisely. Considering the case of supervised representation learning, defining a *fitness* of a latent feature is somewhat possible as there is a clearly defined goal, and latent features can be assessed based on how much they help in achieving the goal. In the case of unsupervised representation learning (URL) [62, 9, 116], in which the obtained representation is not tailored for one specific task but can rather be shared among multiple tasks, there is no such guidance.

To overcome this issue, URL has to resort to different means of evaluating latent features. An interesting view on many of the URL methods is that of *producing latent representation by clustering*. Intuitively, these methods find useful features by compressing the original data by means of a multiple-clustering procedure, in which an instance can belong to more than one cluster. The obtained features indicate cluster assignments of instances and, thus, a classifier learns from cluster memberships instead of the original data. For instance, auto-encoders and Restricted Boltzmann machines typically produce a sparse latent representation where each example has a small number of *activated* latent features. This can be seen as assigning examples to multiple clusterings.

A very influential work in this direction is the work of Coates and Ng [18] that constructs a hierarchy of latent features by repeatedly clustering the data (Figure 5.1). Assuming a spatial order of features (i.e., pixels), the introduced framework (i) extracts image patches, i.e., subsets of pixels from the original images candidating as a potential high-level feature, (ii) pre-processes each patch (e.g. normalisation), and (iii) learns a feature-mapping by clustering image patches. The authors show that such general procedure with a simple k-means algorithm can perform as well as the specialised algorithms, such as auto-encoders and Restricted Boltzmann machines. An interesting part about this approach is that it does not require any new algorithm, but a simple adaptation of the method present in machine learning since '60s [135].

This chapter introduces CUR^{2}LED - a *clustering-based unsupervised relational representation learning with explicit distributed representation*, inspired by the work of [18]. The core idea behind CUR^{2}LED is that capturing groups of similar relational objects serves as a good *proxy* for useful representations. All members in such a group are *locally symmetric* to each other, meaning that for the purpose of reasoning they can be interchanged with one another within such local region. They are approximately symmetric as they are similar, but not identical.

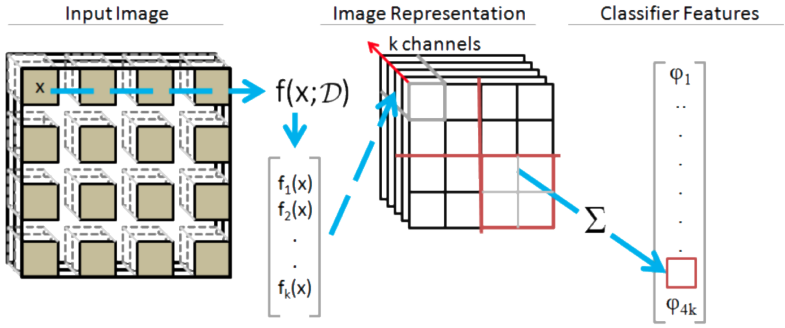


Figure 5.1: **Learning a feature hierarchy with k-means.** The example images are first divided into *patches*. The image patches are then clustered and centroids of the obtained clusters are used as latent features.

Though CUR^{2}LED is not the first approach to rely on a clustering procedure to enhance the predictive performance, it introduces three distinctive features. First, it learns a new representation by clustering both instances and their relationships. Second, the relational structure is preserved throughout the hierarchy, contrasted to the KGE approaches that replace relational structures with vectors. Third, it relies on the notion of *similarity interpretation*. When clustering relational data, a similarity of relational objects is an ambiguous concept. Two relational objects might be similar according to their attributes, relationships, or a combination of both. The notion of similarity interpretation precisely states the exact source of similarity used.

CUR^{2}LED exploits this ambiguity to its advantage by using the similarity interpretations to encode a *distributed representation* of data – one of the pillars underlying the success of representation learning methods. Intuitively, it refers to a concept of reasonably-sized representation that captures a huge number of possible configurations [6]. In contrast to the one-hot representations which require N parameters to represent N regions, distributed representations require N parameters to represent up to 2^N regions. The main difference is that a concept within a distributed representation is represented with several independently manipulated factors, instead of exactly one factor as with one-hot representations. Thus, such representations are substantially more expressive. The similarity interpretation defines the exact factors that can be manipulated individually to represent individual concepts.

The contributions of this work include

(i) a general framework for learning relational feature hierarchies by means of clustering

(ii) a principled way of generating distributed relational representations based on different similarity interpretations

(iii) a general framework for hyper-edge clustering, and

(iv) the experimental evaluation of the proposed framework.

5.2 Representation Learning via Clustering

The complexity of relational data causes several issues in devising a general relational feature hierarchy: *(1) what should be clustered, (2) how to estimate a similarity in relational data, and (3) how to choose the structure of a feature hierarchy?*

(1) What should be clustered? In the case of tabular data, the dataset contains only instances and their features, thus, one clusters the instances. However, relational data additionally describes relationships among instances and varies from a single large network of many interconnected entities (a *mega-example*) to a set of many disconnected networks where each network is an example. Ideally, one would address both cases. CUR^{2}LED assumes the same formalism as ReCeNT: relational data is provided as a labelled hyper-graph, where examples form vertices and relations between them form hyper-edges, and does not make a distinction between the above-mentioned cases. Formally said, the data structure is a typed, labelled hyper-graph $H = (V, E, \tau, \lambda)$ with V being a set of vertices, E a set of hyper-edges, τ a function assigning a type to each vertex and hyper-edge, and λ a function assigning a vector of values to each vertex. CUR^{2}LED learns a new representation by clustering both *vertices and hyper-edges in a hyper-graph*. Considering that vertices have associated types, CUR^{2}LED does not allow mixing of types, i.e., a cluster can contain only vertices of the same type. The same holds for hyper-edges which connect vertices of different types.

(2) How to estimate the similarity? The features are the only source of similarity between instances in the i.i.d. data. In the relational context, a similarity is an ambiguous notion that can originate in the features of relational objects, structures of their neighbourhoods (both feature- and relationship-wise), inter-connectivity or graph proximity, just to name a few. Furthermore, which interpretation is needed for a particular task is not known in advance,

making URL inherently more difficult. To find a representation effective for many tasks, CUR^{2}LED addresses *multiple interpretations of relational similarity simultaneously.* How exactly that is achieved is discussed in the next section.

(3) How to choose the structure? Defining a feature hierarchy requires the specification of the number of layers and the number of hidden features (i.e., clusters) within each layer. How to automatically construct such hierarchies is currently under-explored. Consequently, the performance of these methods is sensitive to the parameter setting, requiring substantial expertise in order to choose the optimal number of features. This constitutes a major bottleneck for relational *type-aware* feature hierarchies, as separate values should be chosen for each type in data (and combination thereof for hyper-edges).

To tackle this infeasibility, CUR^{2}LED builds upon a vast literature on *the clustering selection problem* [2], which is concerned with the selection of the optimal number of clusters from data. This automatic clustering selection strategies mitigate the problem of the manual specification of feature hierarchies. CUR^{2}LED leverages two distinct approaches: *(1) a difference-like criterion* [149], and *(2) a quality based criterion of Silhouette index* [124].

Difference-like criteria assess relative improvements on some relevant characteristic of the data (e.g. within-cluster similarity) over a set of successive data partitions produced by gradually increasing the number of clusters (N). It attempts to identify a prominent *knee* - a point when the given quality measure *saturates* and the further increase of N can offer only marginal benefit. Following the suggestion in [149], we choose the number of clusters as the one that achieves the highest value of the following formula:

$$D(k) = \left| \frac{C(k-1) - C(k)}{C(k) - C(k+1)} \right| - \alpha \cdot k \tag{5.1}$$

where $C(k)$ is the intra-cluster similarity, k is the number of clusters and α a user-specified penalty on the number of clusters.

The Silhouette index evaluates a *cohesion*, i.e., how similar an instance is to its own cluster, and a *separation*, i.e., how similar an instance is to the other clusters. It is defined as:

$$S(i) = \frac{a(i) - b(i)}{\max\{a(i), b(i)\}} \tag{5.2}$$

where i is an instance, $a(i)$ is an average dissimilarity of i to the rest of the instances in the same cluster, and $b(i)$ the lowest dissimilarity of i to any other cluster. Higher values indicate a better fit of the data.

$\mathcal{C}$-representation. Once the clusters are obtained, we will represent them in the following form. For each cluster of vertices we create a unary predicate in the form of `cluster`ID`(vertex)` where `vertex` is an identifier of a specific vertex. Similarly, for each cluster of hyper-edges we create a n-ary predicate in the form of `cluster`ID`(vertex`$_1$`,...,vertex`$_n$`)`, which takes an ordered set of n vertices as arguments. Truth instantiations of defined predicates reflect cluster memberships. We refer to the cluster-induced representation as a $\mathcal{C}$-representation.

The introduced pipeline is illustrated in Figure 5.2. CUR^{2}LED specified thus far describes a *meta-procedure* how to use any clustering algorithm to obtain a latent representation. In the experiments we use spectral and hierarchical clustering.

5.3 Similarity of Relational Structures

CUR^{2}LED relies on the previously introduced approach ReCeNT. What makes ReCeNT an attractive relational clustering framework is the wide range of similarities it considers. We will refer to them as *core similarities*. Furthermore, which similarity is used is easily adaptable with just a few parameters.

5.3.1 Hyper-edge Similarity

In its original form, ReCeNT clusters vertices in a hyper-graph. To support hyper-edge clustering with CUR^{2}LED, we introduce a general framework for hyper-edge similarity. It views hyper-edges as ordered sets of vertices, and thus ordered sets of neighbourhood trees (NTs).

Let $\mathcal{N}$ be a set of NTs. Let Θ denote summary operations on sets of values such as mean, minimum and maximum. Let Λ denote set operators such as union and intersection. Let $f : \mathcal{N}^2 \to \mathbb{R}$ be a similarity between two NTs, e.g. the similarity measure introduced by ReCeNT. The framework introduces two types of hyper-edge similarity, namely *combination* and *merging*.

Definition 5.3.1. *A combination similarity is a function* $c : \mathcal{N}^n \times \mathcal{N}^n \times \Theta \to \mathbb{R}$ *which compares two hyper-edges,* $e_1 = (v_1^1, ..., v_1^n)$ *and* $e_1 = (v_2^1, ..., v_2^n)$*, by*

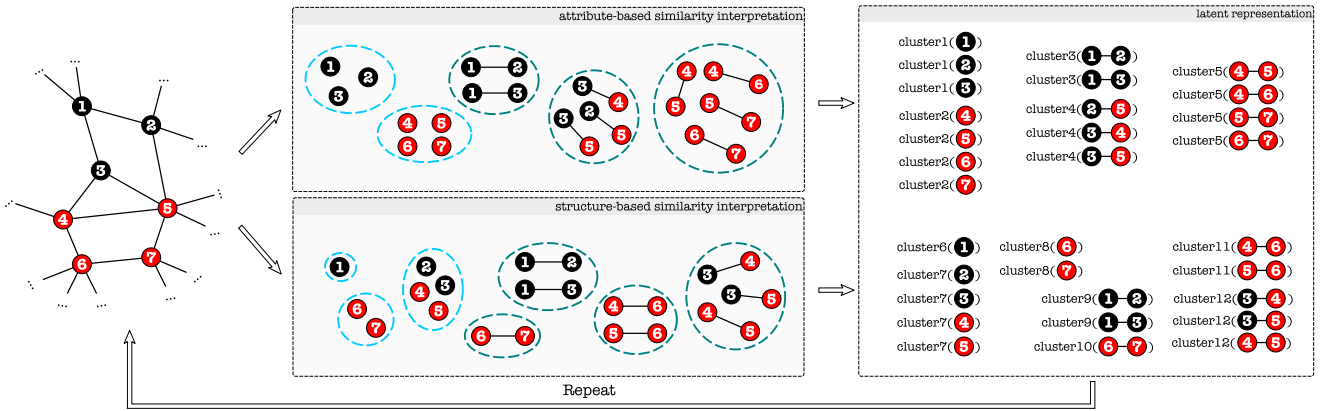
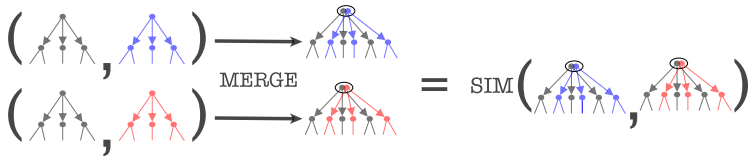


Figure 5.2: **An illustration of CUR^{2}LED procedure.** The left-most figure represents a given hyper-graph, where colour of a vertex indicates its feature value. The graph (i.e., vertices and edges) is then clustered according to different similarity interpretations. The upper clustering is based on vertex attributes: the vertices are clustered into *red* and *black* ones, while the edges are clustered according to the colour of the vertices they connect. The bottom clustering is based on the structure of the neighbourhoods. The vertices are clustered into a group that have only *black* neighbours ({1}), only *red* neighbours ({6,7}), and neighbours of both colours ({2,3,4,5}). The edges are clustered into a group of edges connecting *black* vertices with only *black* neighbours and *black* vertices with *red* neighbours ({1-2,1-3}), a group of edges connecting *red* vertices with only *red* neighbours to *red* vertices with neighbours of both colour ({6-7}), and so on. The final step transforms the obtained clusterings into a relational representation. The procedure can further be repeated to create more layers of features.



(a) Merging neighbourhood trees



(b) Combining neighbourhood trees

Figure 5.3: Illustration of hyper-edge similarity operators

comparing the individual NTs respecting the order, $s = \big(f(v_1^1, v_2^1), \ldots, f(v_1^i, v_2^i)\big)$*, and summarising respective similarities with* $\theta \in \Theta$*,* $\theta(s)$*.*

Definition 5.3.2. *A merging similarity is a function* $m : 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \Lambda \to \mathbb{R}$ *which compares two hyper-edges,* $e_1 = (v_1^1, ..., v_1^n)$ *and* $e_1 = (v_2^1, ..., v_2^n)$*, by first merging the NTs within a hyper-edge with merging operator* $\lambda \in \Lambda$*,* $s_1 = \lambda(v_1^1, \ldots, v_1^i)$ *and* $s_2 = \lambda(v_2^1, \ldots, v_2^i)$ *and comparing the resulting glsnts,* $f(s_1, s_2)$*.*

Merging two NTs involves merging their respecting multi-sets with a merging operator λ, respecting the level. For instance, consider *set union* as λ, and g' and g'' as the NTs to be merged. Then, merging the multi-sets $V_t^l(g')$ and $V_t^l(g'')$ results in a multi-set $V_t^l\ (\lambda(g', g'')) = V_t^l(g') \cup V_t^l(g'')$.

Both formulations reduce the problem to the comparison of NTs, but offer alternative views. While *merging* ignores the order of vertices in a hyper-edge, *combination* respects it. Accordingly, *merging* describes the neighbourhood of a hyper-edge, while *combination* examines the similarity of vertices participating in a hyper-edge. In this work we use *union* as the merging operator, and *mean* as the combination operator.

5.3.2 Similarity Interpretation

Finally, we formally introduce the notion of similarity interpretation.

Definition 5.3.3. *Let* $(w_1, w_2, w_3, w_4, w_5)$ *be the weights associated with the core similarities. A similarity interpretation is the value assignments to the weights* $(w_1, w_2, w_3, w_4, w_5)$.

Thus, it allows us to precisely control aspects of similarity considered for representation learning. For example, setting $w_1 = 1, w_{2,3,4,5} = 0$ uses only the attributes of vertices for comparison. Setting $w_3 = 1, w_{1,2,4,5} = 0$ on the other hand would identify clusters as a densely connected components. As the similarity interpretation is provided by the user, we say it *explicitly* defines the distributed representation.

5.4 Related Work

Clustering has been previously recognised as an effective way of enhancing relational learners. [112] apply k-means clustering to instances, create predicates for new clusters and add them to the original data. *Multiple relational clustering (MRC)* [76, 77] is a relational probabilistic clustering framework based on Markov logic networks [118] clustering both vertices and relationships. Both approaches are instances of predicate invention [81, 20], concerned with extending the vocabulary given to a learner by discovering novel concepts in data. CUR^{2}LED differs in several ways. Whereas [112] develop a method specifically for document classification, CUR^{2}LED is a general *off-the-shelf* procedure that can be applied to any relational domain. Moreover, CUR^{2}LED clusters both instances and relations, whereas [112] cluster only instances. In contrast to MRC which does not put any assumptions in the model, CUR^{2}LED is a more informed approach that explicitly defines different notions of relational similarity to be used for clustering. Though MRC was used as a component in structure learning, it does not provide new language constructs, but simplifies the search over possible formulas. CUR^{2}LED learns a model directly from the new features.

A related problem is community detection [69, 48] concerned with identifying a densely connected components in graphs. CUR^{2}LED considers a more general problem in which *disconnected* vertices (and edges) can form clusters as well.

On contrast to the various KGE methods that invent an Euclidean space of relational instances, CUR^{2}LED relies on clustering and a variety of similarity measures to create new features.

5.5 Experiments and Results

Datasets. We have used the following six datasets to evaluate the potential of this approach. The IMDB dataset describes a set of movies with people acting in or directing them. The UW-CSE dataset describes the interactions of employees at the University of Washington and their roles, publications and the courses they teach. The Mutagenesis dataset describes chemical compounds and atoms they consist of. The WebKB dataset consists of pages and links collected from the Cornell University's web page. The Terrorists dataset describes terrorist attacks each assigned one of 6 labels indicating the type of the attack. The Hepatitis dataset describes a set of patients with hepatitis types B and C.

Evaluation procedure. In principle, a latent representation should make learning easier by capturing complex dependencies in data more explicitly. Though that is difficult to formalise, a consequence should be that a model learned on the latent representation is (i) *less complex*, and (ii) possibly *performs better*. To verify whether that is the case with the representation created by CUR^{2}LED, we answer the following questions:

(Q1) *do representations learned by CUR^{2}LED induce models of lower complexity compared to the ones induced on the original representation?*

(Q2) *if the original data representation is sufficient to solve a task efficiently, does $\mathcal{C}$-representation preserves the relevant information?*

(Q3) *if the original data representation is not sufficient to solve the task, does a $\mathcal{C}$-representation improve the performance of a relational classifier?*

(Q4) *can the appropriate parameters for a specific dataset be found by the model selection?*

(Q5) *how does CUR^{2}LED compare to MRC, which is the closest related work?*

In order to answer these questions, we compare the performances of a relational decision tree learner TILDE [13], and perform leave-one-out cross validation. When using the original data representation, we learn TILDE on training data and test it on the test fold. When using $\mathcal{C}$-representations, we first perform CUR^{2}LED procedure on the training data and learn TILDE on the obtained $\mathcal{C}$-representation. For test, we first map the test fold to the corresponding $\mathcal{C}$-representation by assigning objects and relationships to their closest clusters in each of the similarity interpretations. The following similarity interpretation were used for each dataset: (0.5,0.5,0.0,0.0,0.0), (0.0,0.0,0.33,0.33,0.34), (0.2,0.2,0.2,0.2,0.2). The first set of weights uses only the attribute information, the second one only the link information, while the last one combines every component.

Table 5.1: **Performance comparison for TILDE models learned on the original and $\mathcal{C}$-representations.** The first column specifies the parameters used for $\mathcal{C}$-representation, i.e., clustering algorithm (S-spectral, H-hierarchical), selection criterion and its parameter values. Both accuracies on a test set (Acc) and complexities (Cplx) are reported.

	Setup	IMDB		UWCSE		Mutagenesis		Terrorists		Hepatitis		WebKB	
		Acc	Cplx	Acc	Cplx	Acc	Cplx	Acc	Cplx	Acc	Cplx	Acc	Cplx
	Original	1.0	2.0	0.99	3.0	0.76	27.2	**0.72**	86.4	0.81	22.4	0.81	18.2
merging	S,$\alpha = 0.01$	1.0	1.0	0.99	1.2	0.79	6.6	**0.71**	34.4	0.86	19.66	**0.89**	13.6
	S,$\alpha = 0.05$	1.0	1.0	0.99	**1.0**	0.78	2.4	0.65	21.6	0.90	7.6	0.85	15.6
	S,$\alpha = 0.1$	1.0	1.0	0.99	1.2	0.78	**1.8**	0.66	32.4	0.90	6.5	**0.87**	17.8
	S,silhouette	1.0	1.0	0.99	**1.0**	0.78	**2.0**	0.6	23.6	**0.93**	**5.33**	**0.87**	14.8
	H,$\alpha = 0.01$	1.0	1.0	0.98	4.4	**0.83**	**2.0**	0.48	**9.4**	0.86	12.0	0.83	12.6
	H,$\alpha = 0.05$	1.0	1.0	0.99	4.2	**0.83**	**2.0**	0.48	11.6	0.82	16.0	0.69	27.2
	H,$\alpha = 0.1$	1.0	1.0	0.99	4.0	0.79	5.2	0.47	**8.8**	0.82	13.4	0.61	32.2
	H,silhouette	1.0	1.0	0.98	**1.0**	0.80	3.4	0.47	13.0	**0.93**	8.66	0.68	18.0
combination	S,$\alpha = 0.01$	1.0	1.0	0.99	1.2	0.79	**2.0**	**0.72**	24.0	0.90	7.6	**0.90**	11.8
	S,$\alpha = 0.05$	1.0	1.0	0.99	**1.0**	0.79	**2.0**	0.69	22.8	0.88	12.2	0.86	**10.0**
	S,$\alpha = 0.1$	1.0	1.0	1.0	**1.0**	0.76	**2.0**	0.66	16.8	0.90	12.6	0.87	17.0
	S,silhouette	1.0	1.0	0.99	**1.0**	0.77	**2.0**	0.6	24.2	**0.93**	16.4	**0.88**	13.8
	H,$\alpha = 0.01$	1.0	1.0	0.99	2.8	0.79	4.0	0.51	30.6	0.80	29.33	0.83	12.6
	H,$\alpha = 0.05$	1.0	1.0	0.99	2.8	0.78	2.8	0.51	30.6	0.82	16.33	0.69	27.2
	H,$\alpha = 0.1$	1.0	1.0	0.99	2.8	0.78	11.0	0.50	27.3	0.78	14.0	0.61	32.2
	H,silhouette	1.0	1.0	0.99	2.0	0.80	4.0	0.50	30.0	0.83	11.6	0.68	18.0
MRC	$\lambda = -1$	1.0	1.0	0.93	21.0	0.6	0	0.64	138.7	0.61	99.4	0.64	44.4
	$\lambda = -5$	1.0	1.0	0.95	25.9	0.63	23.5	0.50	126.5	0.84	64.8	0.68	40.0
	$\lambda = -10$	1.0	1.0	0.96	13.7	0.72	35.0	0.51	102.1	0.57	5.7	0.66	40.8

As a complexity measure of a model we report the average number of nodes a trained TILDE model has. We use the following values for the α parameter in Equation 5.1: $\{0.1, 0.05, 0.01\}$. In the case of MRC, we used the following values for the λ parameter: $\{-1, -5, -10\}$. The λ parameter has the same role as α in the proposed approach, affecting the number of clusters chosen for each type.[1]

Results

To answer the above mentioned questions, we perform two types of experiments. Table 5.1 summarises the results of cross validation. The accuracies on test set and the complexities of TILDE models are stated for both original and $\mathcal{C}$-representations. Table 5.2 summarises the results of the model selection where we dedicate one fold as a *validation set*, and perform the

[1]Note that it is difficult to exactly match the values of α and λ as both methods operate on different scales, and the authors do not provide a way how to choose an appropriate value

cross validation on the remaining folds to identify the best parameter values (i.e., the choice of a clustering algorithm, a clustering selection procedure and the appropriate hyper-edge similarity) for each dataset.

Q1. Table 5.1 shows that the models learned on $\mathcal{C}$-representation consistently have lower complexity than the ones learned on the original data. That is especially the case when $\mathcal{C}$-representation is obtained by spectral clustering, which consistently results in a model of a lower complexity. The reduction of complexity can even be surprisingly substantial, for instance on the Mutagenesis and Hepatitis datasets where the model complexities are reduced by factors of 10 and 4, respectively. When the $\mathcal{C}$-representation is obtained with hierarchical clustering, models of lower complexity are obtained on all datasets except the WebKB and UWCSE datasets. These results suggest that the $\mathcal{C}$-representation in general makes complex dependencies easier to detect and express.

Q2. The IMDB and UWCSE datasets are considered as *easy* relational datasets, where the classes are separable by a single attribute or a relationship. Thus, TILDE is able to achieve almost perfect performance with the original data. The original representation is therefore sufficient to solve the task, and we are interested whether the relevant information will be preserved within the $\mathcal{C}$-representation. The results in Table 5.1 do suggest so, as TILDE achieves identical performance regardless of the representation.

Q3. The remaining datasets are more difficult than the previously discussed ones. On the Mutagenesis, Hepatitis and WebKB datasets, $\mathcal{C}$-representation improves the performance. On the Terrorists dataset, however, no improvement in performance is observed. What distinguishes this dataset from the others is that it contains only two edge types (indicating co-located attack, or ones organised by the same organisation), an abundant number of features, while other datasets are substantially more interconnected. Thus, focusing on the relational information is not as beneficial as the features themselves.

These results suggest that $\mathcal{C}$-representations indeed improve performance of the classifier, compared to the one learned on the original data representation. First, the $\mathcal{C}$-representation created with spectral clustering consistently performs better on all datasets, except the Terrorists one. Second, if the learning task does not have a strong relational component, then $\mathcal{C}$-representations are not beneficial and can even hurt the performance (this is the case with the Terrorists dataset). Third, the choice of a clustering algorithm matters, and spectral clustering does a better job in our experiments - it always results in improved or at least equally good performance. Fourth, the choice of treating hyper-edges as ordered (by *combination*) or un-ordered (*merging*) sets is data-dependent, and the difference in performance is observed.

Table 5.2: **Model selection results.** For each dataset, a selected parameters are reported together with the accuracies on the training and test sets. The first element indicates the selected clustering algorithm (S-spectral, H-hierarchical), the second one the clustering selection criteria, while the last one indicates the hyper-edge similarity (C-combination, M-merging). The last column indicates the performance on the original data representation.

Dataset	Parameters	Training	Validation	Original
IMDB	all	1.0	**1.0**	**1.0**
UWCSE	S, silhouette, C	0.99	**1.0**	0.99
Mutagenesis	H, α=0.01, M	0.86	**0.84**	0.79
Hepatitis	S, silhouette, M	0.92	**0.89**	0.8
WebKB	S, α=0.01, C	0.88	**0.88**	0.79
Terrorists	S,α=0.01, C	0.70	0.69	**0.71**

Combining the results from **Q1**, **Q2** and **Q3** shows that *the main benefit* of CUR^{2}LED is *the transformation of data such that it becomes easier to express complex dependencies*. Consequently, the obtained models have lower complexities and their performance often improves.

Q4. To ensure that the previously discussed results do not over-fit the data, we additionally perform model selection. We dedicate one fold as the *validation set*, and use the remaining folds to find the best parameter values of both CUR^{2}LED and TILDE. Table 5.2 summarises the results and reports the selected choice of parameter, together with the performance on the validation set. These results are consistent with the ones in Table 5.1: $\mathcal{C}$-representation improves the performance in the majority of cases, and the selected parameters correspond to the best performing ones in Table 5.1.

Q5. Table 5.1 shows that CUR^{2}LED substantially outperforms MRC on all datasets, achieving better performance on all datasets except the IMDB. Moreover, MRC rarely shows benefit over the original data representation, with an exception on the Hepatitis dataset. Considering the model complexity, the models learned on MRC-induced representation are substantially more complex than the ones learned on $\mathcal{C}$-representations. Table 5.3 summarise the number of clusters created by CUR^{2}LED and MRC. MRC creates substantially more clusters than CUR^{2}LED. Because of this, the found clusters contain only a few objects which makes it difficult to generalise well, and increases the model complexity. The number of clusters found by CUR^{2}LED is relatively high, because it finds a representation of data suitable for many classification tasks over the same datasets. Thus, most of the features are redundant for one specific task, but clearly contain better information as the models learned on them perform better and have lower complexity.

Table 5.3: Number of clusters in various $\mathcal{C}$-representations. M indicates MRC, while S and H indicate CUR^{2}LED representations with spectral and hierarchical clustering, respectively. Vocabulary sizes obtained with *merging* and *combination* similarities were similar, so only the one for merging is reported.

Setup	UW	Muta	WebKB	Terror	IMDB	Hepa
Original	10	12	775	107	5	22
S, $\alpha = 0.01$	109	53	65	30	75	85
S, $\alpha = 0.05$	87	37	63	26	69	66
S, $\alpha = 0.1$	72	31	57	24	59	28
S, silhouette	93	17	59	37	74	79
H, α=0.01	93	38	64	25	69	62
H, α=0.05	85	34	64	20	65	50
H, $\alpha = 0.1$	68	22	58	18	55	46
H, silhouette	85	20	55	43	64	61
M, λ=-1	183	535	817	318	49	655
M, λ=-5	140	346	331	116	38	297
M, λ=-10	49	224	219	91	18	120

The computational complexity of CUR^{2}LED is impossible to state in general, as it depends on the choice of (and is dominated by) the clustering algorithm. Performing a 5-fold cross validation on a single CPU took approximately 5 minutes for the IMDB and UWCSE datasets, 24 hours for the Terrorists dataset and approximately a week for the remaining datasets. Though expensive, latent representation has to be created only once and can be reused for many tasks with the same dataset. Moreover, CUR^{2}LED is easily parallelised which can substantially improve its efficiency.

5.6 Opening the black box of latent features

We have previously argued that the lack of interpretability of the latent features is one of the strongest weaknesses of deep learning methods. In this section we describe a simple method helping us understand the meaning of the latent features created by CUR^{2}LED.

5.6.1 Discovering the explanations of latent features

Although the latent features discovered by CUR^{2}LED are defined extensionally, the intuitive specification of the similarity measure (and its core similarities) makes CUR^{2}LED a transparent method with a clear description at which elements in the NTs make two instances similar. Consequently, discovering the meaning of latent features is substantially easier than with the embedding approaches (and deep learning in general).

Each latent feature corresponds to a cluster and the meaning of the features is reflected in the *prototype* of the cluster. To approximate the *mean* or *prototypical* NT, we search for the elements common to all NTs forming a cluster. These elements can be either attribute values, edge types or vertex identities. The similarity interpretations used to obtain the cluster limits which elements are considered to be a part of a definition. Moreover, NTs are compared by the relative frequencies of their elements, not the existence only. Therefore, to find a mean neighbourhood tree and the meaning of a latent feature, we search for *the elements with similar relative frequencies within each NT forming a cluster*.

To identify such elements, we proceed in three steps illustrated in Figure 5.4.

1. **Calculate the relative frequencies of all elements within each individual neighbourhood tree, per level and vertex type**. In case of discrete attributes, that corresponds to a distribution of its values. In case of numerical attributes, we consider its mean value. In case of vertex identities and edge types, we simply look into their frequencies with respect to the depth in a neighbourhood tree. In the example in Figure 5.4, the neighbourhood tree for `profA` contains two `advisedBy` relations, thus its frequency is $\frac{2}{3}$.
2. **Calculate the mean and standard deviation of relative frequency for each element within a cluster**. In Figure 5.4, the frequencies of the `advisedBy` elements in individual neighbourhood trees are $\frac{2}{3}$, $\frac{2}{3}$and$\frac{1}{3}$. Thus, its mean is 0.55 with a standard deviation of 0.15.
3. **Select relevant elements.** The final step involves deciding which elements should form a definition of a latent feature. Relevant elements are identified by a notion of θ*-confidence* which captures the allowed amount of variance in order to element to be relevant.

Definition 5.6.1. ***(θ-confidence)*** *An element with mean value μ and standard deviation σ in a cluster, is said to be θ-confident if $\sigma \in [0, \theta \cdot \mu]$.*

In Figure 5.4, setting θ to 0.3 makes `advisedBy` a 0.3-confident element, because its standard deviation of 0.15 is within the range $[0, 0.3 \cdot 0.55] = [0, 0.165]$

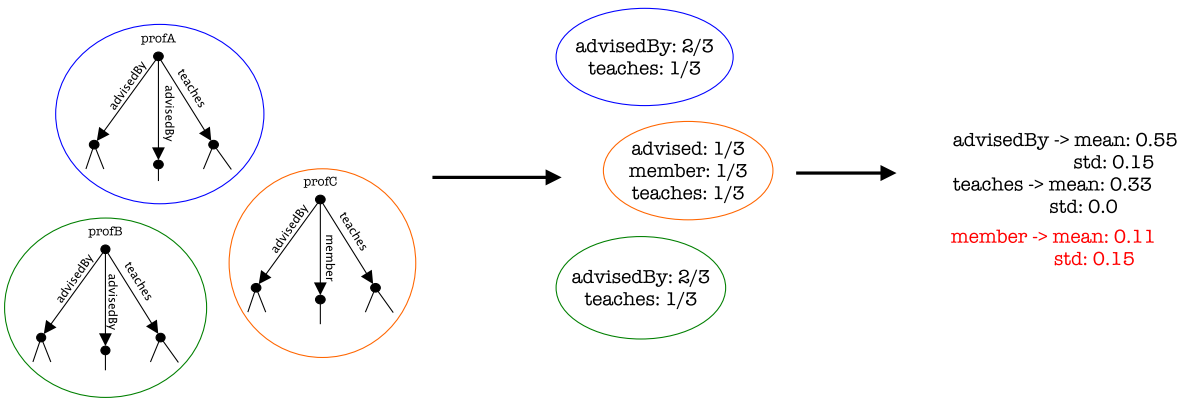


Figure 5.4: **Discovering the meaning of latent features by analysing their relations.** Properties that describe latent features are the ones that have similar relative frequency in all neighbourhood trees. Starting from a cluster of instances viewed as neighbourhood trees (left), the relative frequencies of elements are calculated for each neighbourhood tree (middle). Next, the mean and standard deviation of relative frequencies are calculated for each individual element within the cluster (right). Which elements *explain* the latent features is decided with θ-confidence. Setting θ to 0.3 identifies `advisedBy` and `teaches` as relevant elements (in black).

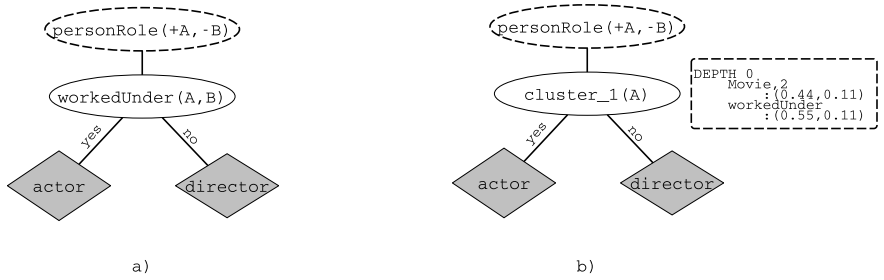


Figure 5.5: Relational decision trees learned on the original (left) and latent (right) data representation of the IMDB dataset. The dashed ellipse indicates the target predicate and its arguments. The first argument, marked `A` and declared as input (+), denotes a person. The second argument, marked `B` and declared as output (-), states the label of the instance given by `A`. The values in the leaves of the decision trees are assignments to `B`. The dashed rectangle describes the latent feature – for each level of the *mean neighbourhood tree*, θ-confident elements are listed with the mean and standard deviation.

specified by θ. In contrast, `member` is not a 0.3-confident elements as its standard deviation is outside the range $[0, 0.3 \cdot 0.11] = [0, 0.0363]$.

The above-described procedure explains the latent features in terms of distribution of the elements in the neighbourhood of an instance, which has its pros and cons. On the downside, this type of explanation does not conform to the standard first-order logic syntax common within relational learning. Despite this reduced readability, these explanations are more transparent and readable compared to the deep neural networks. However, one benefit of this approach is that it increases the expressivity of a relational learner by extensionally defining properties otherwise inexpressible in the first-order logic.

5.6.2 A few examples

To illustrate the interpretability of relational features, we show examples of latent features created for two easy to interpret dataset - IMDB and UWCSE. We show that the relational decision trees learned on both original and latent representations. The explanations of latent features are provided as well.

Figure 5.5 shows the decision trees learned on the IMDB dataset. The task is to distinguish between actors and directors – this is a simple relational

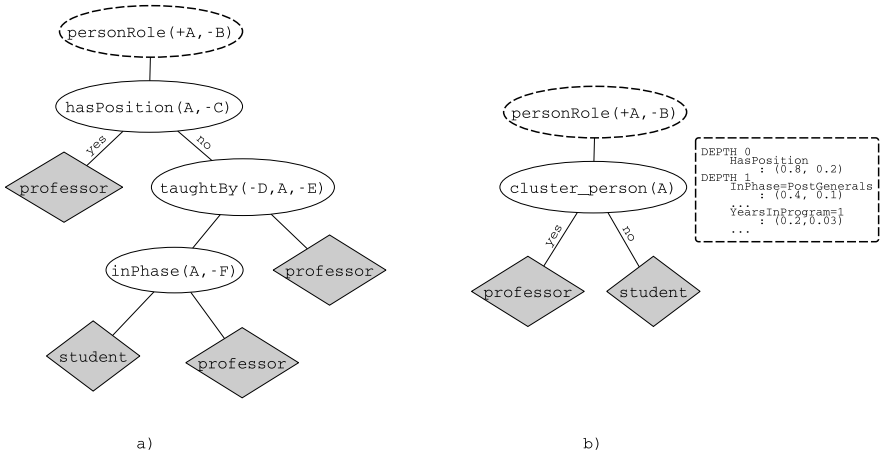


Figure 5.6: Relational decision trees learned on the original (left) and latent (right) representations of the UWCSE dataset. The elements have the same meanings as in Figure 5.5.

learning task and both original and latent decision tree achieve the perfect performance with only a single node. Even though latent representation does not seem beneficial in this particular case, it is interesting to see that the selected latent feature captures the same information as the decision tree learned on the original data – person instances in `cluster_1` are the ones that have a relationship with `movie` instances, and have worked under another person (a director).

Figure 5.6 shows the decision trees for the UWCSE dataset, which benefit from the latent features. Despite the simplicity of distinguishing students from professors, the decision tree learned on the latent features is more compact and has only a single node whereas the decision tree learned on the original features consists of three nodes. The latent feature here again captures similar knowledge as the original decision tree but expressed in a simpler manner – professor is someone who either has a position at the faculty, or is connected to people who are currently in a certain phase of a study program and have been in the program for a certain number of years.

What is particularly interesting about the examples above is that, even though the latent features are created in an unsupervised manner, they match the provided label very well. Moreover, they seem to almost perfectly capture the labelled information as only a few features are needed to outperform the

decision tree learned on the original data representation. This observation shows that CUR^{2}LED is indeed capturing sensible knowledge in the latent space.

Both aforementioned examples are easy to understand and interpret without an extensive domain knowledge. The other tasks that have benefited more from the latent features are substantially more difficult to understand. For instance, the latent features created from the Mutagenesis dataset reduce the complexity of the relational decision tree from 27 to only 3 nodes, while improving the accuracy for 4 %. Similarly, on the Hepatitis dataset the latent features reduced the complexity of a decision tree from 22 nodes down to 5, improving the accuracy for 11 %.

5.7 Conclusion

This chapter has introduces CUR^{2}LED - a clustering-based framework for unsupervised representation learning with relational data, which describes both instances and relationships between them. Viewing relational data as hyper-graph, CUR^{2}LED learns new features by clustering both instances and their relationships. i.e., vertices and hyper-edges in the corresponding hyper-graph. To support such procedure, we introduce a general hyper-edges clustering framework based on similarity of vertices participating in the hyper-edge. A distinct feature of CUR^{2}LED is the way it uses different interpretations of similarity for relational data, i.e., whether two relational objects are similar due to their features of relationships, to generate a new data representation. We design several experiments to verify the usefulness of latent representation generated by CUR^{2}LED. The results show that the latent representations created by CUR^{2}LED provide a better representation of data that results in models of lower complexity and better performance.

Chapter 6

Auto-encoding logic programs

This chapter discussed the third contribution of the thesis in the form of *auto-encoding logic programs*. Auto-encoding logic programs lift the auto-encoding principle from vector representation to logical data formats. More precisely, it uses clauses logic for data representation and as a computational framework for encoder and decoder functions.

The chapter is based on the following publications.

> DUMANČIĆ, S., GUNS, T., MEERT, W., AND BLOCKEEL, H. Auto-encoding logic programs. In *Proceedings of the 2nd Workshop on Neural Abstract Machines and Program Induction* (2018)

> DUMANČIĆ, S., GUNS, T., MEERT, W., AND BLOCKEEL, H. Auto-encoding logic programs. *preparations* (2018)

6.1 Introduction

The previous chapter introduces CUR^{2}LED – a method for relational representation learning that creates latent representation by exploiting symmetries in data. Even though CUR^{2}LED brings concepts from the SRL into relational representation learning, it relies on the external clustering procedure to *invent* new predicates. Thus, the constructed representation hierarchy is not truly relational as latent features are not defined through logical formulas, even though we managed to find a way to explain the meaning of the constructed

features. Additionally, latent representations created by CUR^{2}LED are prone to generating a rather large number of latent features, of which only a small portion is actually used in the subsequent predictive model. Though the discovered features are likely to be useful for many different tasks, they make the generative learning of SRL models difficult as a large number of predicates results in a large search space.

In this chapter we ask the following question: *how can we learn latent representations of relational data in a principled framework that can keep data and reasoning at the relational level?* We are interested in learning relational latent representation by means of learning and inference methods from the SRL community.

We argue that fully retaining first-order logic as a data representation language within deep learning approaches offers several benefits. First, predicate logic is a Turing-equivalent programming language and thus provides a language that does not introduce any limitations upfront. Second, it is easily interpretable (whereas DL is a *black-box*) and modular which might allow for efficient learning strategies. Third, once the domain model is obtained, SRL systems can answer any question w.r.t. the provided domain model and the user thus does not have to commit to a specific learning task in advance. Fourth, SRL and other logic-based ML methods are very data-efficient and capable of learning from a few examples only, whereas DL is *data-hungry*. Fifth, SRL methods allow for incorporation of *background knowledge* and thus can easily build further on previously gathered knowledge.

This work takes a fresh look at the representation learning problem and induces relational latent representations in a *symbolic way*, instead of the gradient-based optimisation. We introduce *auto-encoding logic programs* (Alps) – a novel relational DL component which uses logic programs as a computation engine instead of neural networks. We focus on auto-encoders as they have proven to be among the most versatile DL primitives and applicable to different learning settings (un-, semi- and supervised learning) using the same learning principle [73, 151, 74, 8]. This versatility is important as it makes a single latent representation suitable for many tasks within SRL. There tasks range from learning predictive models to structure learning. A key difference between the predicate invention approaches and Alps is that that we create latent representations in an entirely unsupervised manner, whereas predicate invention is defined as a supervised method. An exception to that is the work related to *statistical predicate invention* [76] which also addresses a generative learning setup within SRL, but never explicitly creates latent representation. Moreover, it is not used to provide novel language constructs to an SRL learner, but only to compress the existing KB by identifying entities *identical* to each other and thus speed up learning [78]. If one would explicitly construct the

latent representation, learning generative models would be extremely difficult due to the very large number of created constructs.

First learning a latent representation can be beneficial for the main learning task of SRL, namely structure learning, in which the goal is to find a set of formulas and the corresponding probabilities. The main reason why (structure) learning in SRL is a difficult problem is that the learning task is in its core a search problem with the aim of identifying a small set of predictive logical formulas. Therefore, it suffers from the combinatorial explosion which comes from a large number of logical formulas that can typically be constructed given a fixed vocabulary. Interpreting candidate formulas as individual features, this is reminiscent of the problem of learning from high-dimensional spaces, which is the problem DL tackles. Therefore, incorporating DL ideas into SRL might reduce the complexity of structure learning and/or increase the quality of the obtained models.

The contributions of this work are a (i) *a framework for learning auto-encoding logic programs* which is both data-driven and data-efficient, interpretable, and unsupervised, which means that the resulting latent representation is suitable for many SRL learning tasks; (ii) *a generic procedure to cast learning the auto-encoders as a constraint optimisation problem, for which existing efficient solvers can be used*; and (iii) *an experimental evaluation of the proposed framework that shows the benefits for generative learning*.

6.2 Auto-encoding Logic Programs

The core idea of auto-encoders is the *reconstruction principle*: the goal is to learn an *encoder*, mapping the provided data to its latent representation, and a *decoder*, reconstructing the original data from the latent space, such that the reconstruction is maximal. The key ingredient of their success is the limited capacity of the latent representation, ensured by limiting its dimensionality and enforcing sparsity. *Traditional* auto-encoders use vectors as a data representation format, and matrices to represent the mapping functions for encoder and decoder.

Intuitively, our goal is to lift the framework of auto-encoders to use *first-order logic as a data representation language*, and *logic programs* as mapping functions of the encoder and decoder (Figure 6.1). It is important to note that by assimilating logic programs, the mapping functions of Alps are realised in a Turing-complete language, giving them the potential to learn arbitrarily complex latent representations.

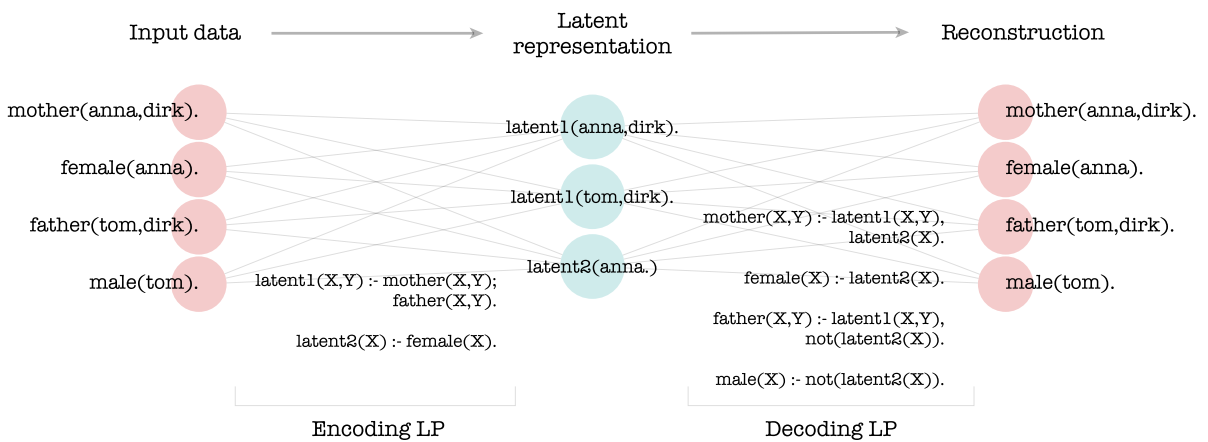


Figure 6.1: Illustration of the auto-encoding logic program learning the concept of *parent* in the latent representation (underlay figure of neural auto-encoder for convenience).

Preliminaries. In order to formalise the learning framework, let $\mathcal{KB}$ be a knowledge base (a set of facts) with a vocabulary $\mathcal{V} = \{\mathcal{P}, \mathcal{C}\}$ where $\mathcal{P}$ is a set of predicates and $\mathcal{C}$ a set of constants. Let $\mathcal{L}$ be a set of predicates not in $\mathcal{P}$; these predicates are called *latent predicates*. Let $\mathcal{KB}_\mathcal{L}$ be a *latent knowledge base* with a vocabulary $\mathcal{V}_\mathcal{L} = \{\mathcal{L}, \mathcal{C}\}$. Let $\mathcal{HB}(\mathcal{P}, \mathcal{C})$ be a *Herbrand base* - a set of all ground atoms composed with $\mathcal{P}$ and $\mathcal{C}$ (equivalently for $\mathcal{HB}(\mathcal{L}, \mathcal{C})$). Then $\mathcal{KB} \subset \mathcal{HB}(\mathcal{P}, \mathcal{C})$ and $\mathcal{KB}_\mathcal{L} \subset \mathcal{HB}(\mathcal{L}, \mathcal{C})$ with a constraint that $\forall e \in \mathcal{KB}, e = \mathit{true}$ (under the *closed-world assumption*, everything not specified as *true* is *false*).

The framework. The individual components of the framework are defined as follows.

Definition 6.2.1. ***Relational encoder.*** *A relational encoder (Figure 6.1, left) is a logic program $\mathcal{E}$ that maps a set of facts $\mathcal{KB} \subset \mathcal{HB}(\mathcal{P}, \mathcal{C})$ to a set of latent facts $\mathcal{KB}_\mathcal{L} \subset \mathcal{HB}(\mathcal{L}, \mathcal{C})$. The clauses of the encoder are termed encoder clauses and their bodies consist of predicates in $\mathcal{P}$, while the heads are composed of predicates in $\mathcal{L}$.*

Definition 6.2.2. ***Relational decoder.*** *A relational decoder (Figure 6.1, right) is a logic program $\mathcal{D}$ that maps a set of latent facts $\mathcal{KB}_\mathcal{L} \subset \mathcal{HB}(\mathcal{L}, \mathcal{C})$ to a new set of facts $\mathcal{KB}' \subset \mathcal{HB}(\mathcal{P}, \mathcal{C})$. The clauses are termed decoder clauses and their bodies consist of predicates in $\mathcal{L}$, while the heads are predicates in $\mathcal{P}$.*

Definition 6.2.3. ***Auto-encoding logic program (Alp).*** *An auto-encoding logic program is a logic program that, given a knowledge base $\mathcal{KB}$, constructs encoder $\mathcal{E}$ and decoder $\mathcal{D}$ programs, together with the latent predicate vocabulary $\mathcal{L}$.*

For defining the loss for the Alps, we differentiate between *true reconstructions* – ground atom that are indicated as *true* in the given knowledge base $\mathcal{KB}$, and *false reconstruction* – ground atoms produced by the decoder but not present in $\mathcal{KB}$.

Definition 6.2.4. ***Knowledge base reconstruction loss.*** *The knowledge base reconstruction loss (the difference of the leftmost and rightmost part of Figure 6.1), $\mathit{loss}(\mathcal{E}, \mathcal{D}, \mathcal{KB})$, is defined as*

$$\mathit{loss}(\mathcal{E}, \mathcal{D}, \mathcal{KB}) = |\mathcal{D}(\mathcal{E}(\mathcal{KB})) \ominus \mathcal{KB}| \tag{6.1}$$

where $\ominus$ is the symmetric difference between two sets (the given knowledge base $\mathcal{KB}$ and the decoded $\mathcal{D}(\mathcal{E}(\mathcal{KB}))$), which returns (1) all non-reconstructed facts from $\mathcal{KB}$ and (2) all false reconstructions.

In defining the task of learning relational auto-encoders we closely follow the learning task of *inductive logic programming*, and make use of *background*

knowledge and *language bias*. Background knowledge represents additional knowledge about the domain a user might have, typically a set of rules, which is not present in the $\mathcal{KB}$ (for instance, a rule saying that every mother is a good cook *good_cook(X) :- mother(X,Y).*). It can be used while constructing the encoder candidates, but is not part of the reconstruction. A language bias is a set of instructions on how to syntactically compose candidate clauses (typically specified using syntactic constraints, e.g., all conjunctive and disjunctive formulas containing at most 3 literals and at most 2, existentially quantified, variables) (see Supplementary material for details). This defines a set of *candidate* encoder and decoder clauses.

Definition 6.2.5. ***Learning Alps. Given*** *a (set of) knowledge base(s) and background knowledge about the domain, a language bias and constraints on the latent representation,* ***find*** *a set of encoder and decoder clauses that minimise Equation 6.1.*

By constraints on latent representation we mean any constraint that affects the interactions of latent predicates; for example, stating that the latent $\mathcal{KB}$ can have at most N facts.

6.3 Learning as constraint optimisation

In this section we show how the above defined problem of learning Alps can be cast as a *constraint optimisation problem* (COP) [123] and provide a general principle how to map the learning problem to a set of constraints. We chose COP as a target solver paradigm as it allows us (1) to deal with large search spaces (current COP technology can deal with millions of variables, which is the scale of DL problems and much larger than traditional logic programs) and (2) to specify the problem declaratively, that is, by describing the problem and including the objective function, but without providing the solving procedure.

Intuitively, the mapping is done in the following way. The language bias provided by the user determines all possible candidate encoder clauses. This is done by specifying all logical formulas that can be used as bodies as encoder clauses, and appending a fresh latent predicate as the head. Having the candidate encoder clauses, all possible decoder clauses are specified in the same manner. The problem now consists of selecting a subset of encoder and decoder clauses that satisfies all constraints and gives the minimal reconstruction loss.

A *constraint optimisation problem* consists of three components: **decision variables** whose values have to be assigned, **constraints** on values and interaction of decision variables, and an **objective function** stating the

preference over the assignments of values to decision variables. We build a COP to learn an Alp as follows.

Decision variables. Each candidate encoder and decoder clause is associated with a *Boolean decision variable* which indicates whether a clause is selected (having the value 1) or not (having the value 0).

Objective function. The objective function captures the reconstruction ability of a certain selection of encoder and decoder clauses by means of Equation 6.1: $loss(\mathcal{E}, \mathcal{D}, \mathcal{KB}) = |\mathcal{D}(\mathcal{E}(\mathcal{KB})) \,\Delta\, \mathcal{KB}|$. To compute this, we have to know which facts in $\mathcal{KB}$ each of the decoder clauses reconstructs, as well as which facts not present in $\mathcal{KB}$ are reconstructed. This can be achieved by first executing the encoder, and using the obtained latent facts to execute the decoder. We can now create Boolean literals representing for all possible facts whether it is (re)constructed by a decoder clause or not.

For instance, assume that the fact `mother(anna,dirk)` can be reconstructed with any of the following two decoder clauses:

```
mother(X,Y) :- latent1(X,Y),latent2(X).

mother(X,Y) :- latent3(X,Y).
```

Assuming that the two clauses correspond to the decision variables $\mathtt{dc}_1$ and $\mathtt{dc}_2$, we add the constraint

$$\mathtt{rf}_i \Leftrightarrow \mathtt{dc}_1 \vee \mathtt{dc}_2$$

where $\mathtt{rf}_i$ is a new Boolean variable propagating the information whether fact `mother(anna,dirk)` is reconstructed or not. $\Leftrightarrow$ is the reification operator, ensuring that $\mathtt{rf}_i$ is assigned to 1 if at least one of the decoder clauses $\mathtt{dc}_1$ and $\mathtt{dc}_2$ is selected.

Using these `rf` variables, we can formulate the objective function by counting how many facts present in $\mathcal{KB}$ are not reconstructed and how many atoms not present in $\mathcal{KB}$ are being reconstructed:

$$\min \underbrace{\sum_{i=1}^{N} \mathrm{not}(\mathtt{rf}_i)}_{\textit{true}\text{ reconstruction}} + \underbrace{\sum_{j=1}^{M} \bar{\mathtt{rf}}_j}_{\textit{false}\text{ reconstruction}} \tag{6.2}$$

where N is the number of facts, and M the number of false reconstructions (indicated with a bar, $\bar{\texttt{rf}}$). The optimisation task then corresponds to finding the assignment to decision variables that minimise the reconstruction loss.

Constraints. Constraints have two primary functions in the proposed framework: (1) imposing the *capacity constraint* which controls sparsity in the latent representation, and (2) imposing structure on the search space to speed up the search for the solution.

Capacity constraints, such as sparsity or compression, are the key ingredient of *traditional auto-encoders* preventing them from learning the *identity mapping* as the latent representation. We impose such a constraint by limiting the *average number of facts per latent predicate* through the following hard constraint

$$\frac{\sum_{i=1}^{N} w_i \texttt{ec}_i}{\sum_{i=1}^{N} \texttt{ec}_i} \leq \gamma G$$

where $\texttt{ec}_i$ are decision variables corresponding to the encoder clauses, w_i indicates the number of latent facts they entail and which can be pre-computed, γ is the *compression parameter* and G is the average number of facts in the original data representation. We have also tried a more straightforward constraint of limiting the number of facts in the latent representation which gives worse results, so we do not consider it further.

The constraint problem specified so far is sufficient to find the latent representation, but search progresses very slowly. To tackle this issue, we impose more structure in the search space by means of the following types of constraints.

Connecting encoder and decoder. A large part of the search space can be cut out by noticing that the encoder clauses deterministically depend on the decoder clauses. This reduces the problem to searching only over decoder clauses, and retaining only the encoder clauses needed to construct selected decoder clauses. To achieve this, we impose constraints of the following two types:

$$\texttt{dc}_k \Rightarrow \texttt{l}_i \ \wedge \ \texttt{l}_j$$

$$\texttt{l}_i \Rightarrow \texttt{dc}_m \ \vee \ \texttt{dc}_n$$

The first constraint states the if a decoder clause $\texttt{dc}_k$ is selected, then encoder clauses $\texttt{l}_i$ and $\texttt{l}_j$ have to be selected as well (assume they are used in the body of $\texttt{dc}_k$). The second constraint state that an encoder clause $\texttt{l}_i$ can be selected only if at least one of the decoder clauses using it in the body, $\texttt{dc}_m$ and $\texttt{dc}_n$, is selected.

Refinement constraints. Given the limited capacity of the latent representation, it is desirable to encourage diversity among the selected latent predicates, and prevent the solver from exploring regions where encoder clauses are too similar (yielding marginal gain). Assume that a clause $\texttt{ec}_1$ is a direct refinement of another clause $\texttt{ec}_2$, i.e., $\texttt{ec}_1$ is obtained by adding a literal to the body of $\texttt{ec}_2$. As $\texttt{ec}_1$ introduces only a subset of atoms reconstructed by $\texttt{ec}_2$, we cannot gain anything by selecting both clauses. Therefore, we introduce the following constraint

$$!(\texttt{ec}_1 \wedge \texttt{ec}_2) == \texttt{true}$$

simply stating that both clauses can not be part of the solution together.

Syntactic variants. A common way to reduce the search space in relational learning is to remove *syntactic variants*. Two encoder clauses are syntactic variants if they have the same head predicate and entail the same facts. Therefore, including a clause that is a syntactic variant of the clause already in the solution does not yield any benefit. However, syntactic variants cannot be simply excluded from the search as some two syntactic variants might depend on different encoder clauses. To ensure only one of the syntactic variants is part of the solution, we impose the following constraint, assuming that $\texttt{dc}_1$,$\texttt{dc}_2$, $\texttt{dc}_3$ are syntactic variants:

$$\texttt{dc}_1 + \texttt{dc}_2 + \texttt{dc}_3 \leq 1.$$

Reconstruct all predicates. Assume that $\texttt{dc}_k$,$\texttt{dc}_l$ and $\texttt{dc}_m$ are decoder clauses all having predicate p as the head predicate. we introduce the following constraint to state that at least one of them has to be selected:

$$\texttt{dc}_1 \vee \texttt{dc}_2 \vee \texttt{dc}_3 == \texttt{true}.$$

We have noticed that this constraint allows the solver to find good solution faster; the reason being that this constraint eliminates the solutions that reconstruct well one predicates with a high number of facts while ignoring other predicates with a lower number of facts.

6.3.1 Search

Given the combinatorial nature of Alps, exact and complete search is impossible in all but the smallest problem instances. Therefore, we resort to a standard technique in constraint programming, namely *large neighbourhood search* (LNS)

[1]. LNS combines complete search with random exploration. It imposes a limit on the number of backtracks for individual runs of a complete search; once the limit is reached, it restarts the search in a random neighbourhood around the current best solution. This is typically done by assigning a small portion of decision variables to their values in the best solution found so far, and searching over the remaining variables.

A key design choice in applying LNS is how to define the neighbourhood. We introduce the following *remember-forget* strategy for constructing neighbourhoods. The strategy is based on noticing that the solution is necessarily sparse - only a tiny proportion of candidate decoder clauses will be a part of the solution at any time. Therefore, when restarting it is important to preserve at least some of the selected decoder clauses from the best solution so far. Assume $\mathcal{B}$ is the assignment of variables corresponding the best solution found so far. Let a variable be *active* if it is selected in $\mathcal{B}$, and *inactive* otherwise. We construct the neighbourhood by *remembering* the value assignments of α % active variables (corresponding to decoder clauses), and *forgetting* (i.e., setting the value to 0) β % of candidate encoder clauses. For the individual search runs, we use *last conflict search* [53] and the *max degree* ordering of decision variables.

6.3.2 Pre-processing

As the candidate clauses are generated purely syntactically, many of the candidate will be uninformative and introduce many false reconstructions. We can further reduce the number of candidates, and hence the search space, through the following pre-processing steps that use specific properties of the problem at hand:

Definition 6.3.1. ***Naming variants.*** *Given two encoder clauses $ec_1(X)$:-... and $ec_2(X)$:-... with L1={$ec_1(a)$,$ec_1(b)$,...} and L2={$ec_2(a)$,$ec_2(b)$, ...} the respective sets of true instantiations. The two clauses are naming variants if renaming the predicate names in L1 and L2 to a common name yields identical sets L1 and L2.*

In order to reduce the search space, we detect all naming variants and keep only one instance as a candidate.

Definition 6.3.2. ***Signature variants.*** *Assume two decoder clauses, $\texttt{dc}_m$ and $\texttt{dc}_n$, with the same head predicate. Let C_m (C_n) be the maximal set of facts decoder clauses $\texttt{dc}_m$ ($\texttt{dc}_n$) entail, and B_m (B_n) be the maximal set of predicates used in the body of the two clauses. The two clauses are signature variants if $C_m = C_n$ and $B_m = B_n$.*

As signature variants are redundant w.r.t. the optimisation problem, we keep only one of the clauses detected to be signature variants and remove the rest.

Definition 6.3.3. ***Corruption level.*** *Given a decoder clause* $\mathtt{dc}_i$*, its true instantiations* $\mathtt{DC}$ *and the original interpretation* $\mathcal{KB}$*, the corruption level is defined as* $c(\mathtt{dc}_i) = \frac{|e \in \mathtt{DC} \wedge e \notin \mathcal{KB}|}{|\mathtt{DC}|}$.

The notion of the *corruption level* turns out to be particularly important, because if $c(\mathtt{dc}_i) \geq 0.5$ then the decoder clause $\mathtt{dc}_i$ cannot improve the objective function as it introduces more *false positives* than *true positives* (under the assumption that the false reconstructions are not shared between the decoder clauses). We remove the candidate clauses that have a corruption level ≥ 0.5.

6.4 Experiments and results

The experiments aim at answering the following question:

> *Does inducing a latent representation with Alps help with solving SRL tasks?*

We focus on evaluation in a generative learning settings, and focus on learning generative *Markov Logic Networks* [118]. This means, more precisely, that we are interested in whether learning the structure of a generative model in the *latent space*, and decoding it back to the original space, is more effective (in terms of how well it explains the data) than learning the model in the original data space.

To answer the question, we apply the following procedure. The baseline MLN model is simply learned on the original data representation, and the average *leave-one-out* AUC-PR is reported. We report AUC-PR instead of AUC-ROC because AUC-PR is less sensitive to the class imbalance problem [25], which is the case with the datasets we use in the experiments. To learn the *latent* MLN model, we first learn the latent representation with Alps varying the length of clauses and the compression level, and learn an MLN on the resulting latent representation. To evaluate the learned MLNs, we query each fact regarding one specific predicate given everything else as evidence. This is repeated for each predicate in a test interpretation. In case of latent MLN, predicate is removed before the rest of the data is encoded to the latent space. To allow for comparing the latent MLN with the baseline model, once the structure is learned in the latent space we add the corresponding decoder clauses from Alps to the latent model. An issue with doing that is that the Alps as specified under the *clausal logic* semantics while MLNs operate under the *first-order logic*

semantics. To convert between the two semantics, we rely on the clausal normal form [147, 125, 65] in which the clause

$$\texttt{h}(\cdot) \texttt{ :- } \texttt{b}_1(\cdot),\texttt{b}_2(\cdot).$$

can be written as the following first-order logic formula

$$\texttt{h}(\cdot) \vee \neg\texttt{b}_1(\cdot) \vee \neg\texttt{b}_2(\cdot).$$

Therefore, we convert each decoder clause in the same way and add it to the MLN as a deterministic formula. This ensures that the baseline and latent MLNs operate in the same space. The only difference when performing inference with latent MLN is that each latent predicate that could have been effected by removal of the test predicate (i.e., the test predicate is present in the body of the encoder clause defining the specific latent predicate) has to be declared *open world*, otherwise MLNs will assume that all atoms not present in the database are *false*.

We limit the expressivity of MLN models to only formulas of length 3 with at most 3 variables (also known as a liftable class of MLNs). This does not sacrifice the performance of MLNs, as shown in [148] which achieves the state-of-the-art structure learning results while imposing the same restrictions to MLNs. We use BUSL [90] as a generative MLN learner.

When learning latent representations, we vary the length of encoder and decoder clauses in $\{2, 3\}$ and the compression level (the α parameter) in $\{0.3, 0.5, 0.7\}$. This choice of parameters expresses a trade-off between long, complex rules and the search space size. Also, this does not exclude learning longer rules since several layers of latent predicates/Alps can be *stacked*. This is a useful property in an SRL setting. Suppose a formula of length 6 is ideal. Instead of searching in a huge space to find such a formula, the latent representation captures short formulas and stacks those to build the longer formula of size 6.

Data We use standard SRL benchmark datasets to evaluate Alps. *WebKB* contains a set of web pages scrapped from four US universities - Cornell and Universities of Texas, Washington and Wisconsin. The data contains web pages, their mutual links and more semantic information about relationships between pages, such as whether the page belongs to a student or a faculty member and courses they teach (we omit the word information, as in [90]). *Cora-ER* contains a set of papers, their authors, title and venues, together entity resolution information (i.e., titles, authors and venues that refer to the same entities but are spelled differently). *UWCSE* contains the information about

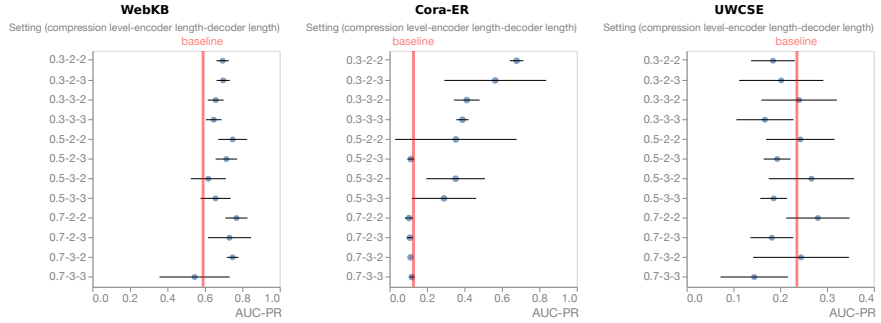


Figure 6.2: The MLN models learned on the latent representation outperform the baseline MLN (indicated with the vertical red line) in terms of the AUC-PR score, except on the UWCSE dataset where the gain is marginal.

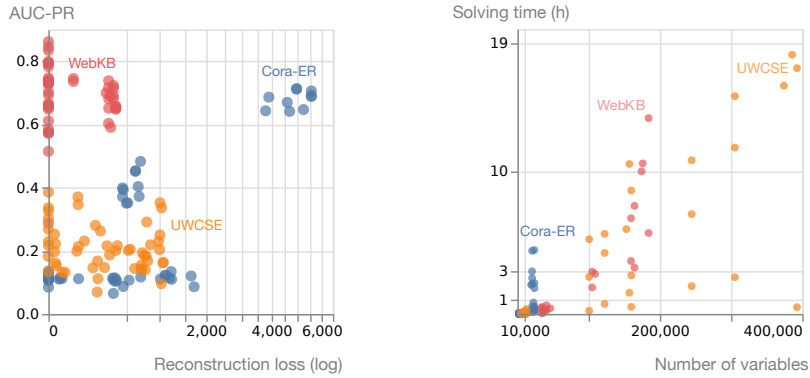
the employees of five departments at the CS department of University of Washington: student and professors together with mentorship relationships, courses they teach and so on. We have experimented with other common SRL datasets, but have removed the on which BUSL did not manage to learn a model (AUC-PR < 0.2) (and IMDB on which no improvement was observed as the dataset is very easy to model).

6.4.1 Results

Figure 6.2 summarises the performance, in terms of AUC-PR scores, of MLNs learned on the original data representation and various representation created by Alps by modifying the clause lengths for both encoder and decoder, and varying the compression level of the latent representations.

The results indicate that latent representation created by Alps are useful tool for relational learning. The improvements are substantial for the WebKB and Cora-ER dataset: on the WebKB dataset the performance improves from ≈ 0.6 to ≈ 0.8, while on the Cora-ER dataset the score improves from ≈ 0.2 to ≈ 0.7. However, the improvement on the UWCSE dataset if marginal – it improves from 0.23 to 0.28.

The results also indicate that the quality of the latent representation depends highly on the chosen hyper-parameters, a property that is observed with the vast majority of representation learning approaches. Whereas on WebKB all setups but one $(0.7 - 3 - 3)$ improve the performance, the Cora-ER and UWCSE tend to be sensitive to the chosen parameters. On Cora-ER, it seems to



(a) Relationship between reconstruction loss and AUC-PR scores.

(b) Relationship between runtimes and the number of variables.

Figure 6.3: Aspects of learning procedure for Alps

be important to compress the data heavily as setups with the compression level of 0.3 perform the best. One possible explanation is that Cora-ER contains a lot of word information; to find useful latent information about words, Alps have to be forced to compress a lot, otherwise they are simply extracting patterns that are too general.

The results for the UWCSE dataset show that the performance gain is marginal, improved by only 0.05 points, and most of the latent MLNs perform *worse* than the baseline MLNs. Upon further investigation, we discovered that marginal improvement is the result of Alps getting stuck in local optima: this dataset has relatively large number of predicates and therefore yielding a very large search space (Figure 6.3b) which contains many local optima. We noticed that these local optima had a particular *cheat*: the encoder was composed of *identity mappings* between the observed and the latent predicates and many specific encoder clauses that entail very small number of latent facts, resulting in having a small *average number of facts per latent predicate* and cheating the capacity constraint. We tried to circumvent this problem by imposing the constraint that prevent two decoder clauses, where facts entailed by one clause are a subset of facts entailed by the other clause, being selected together; however, that was computationally infeasible given the large number of candidate clauses. Some results show that certain setups lead to the perfect reconstruction loss (Figure 6.3a) – this is the result of over-fitting.

It is interesting to see that there is no general trend relating reconstruction loss and performance (Figure 6.3a). On the WebKB, the best performing models are

those with (near) perfect reconstruction; on the Cora-ER, the best performing models seems to have relatively high reconstruction loss; while on the UWCSE, the best performing models seem to occupy both parts of the spectrum. This might indicate the amount of noise present in the data, suggesting that WebKB is a rather clean dataset, while Cora-ER contains quite some noise and the best performing models are learned on latent representation that manage to reduce the amount of noise.

Figure 6.3b summarises the time needed for learning a latent representation. These timings show that, despite being defined as a combinatorial problem, Alps are quite efficient: the majority of latent representations is learned within one hour, and a very few taking more than 10 hours (this excludes the time needed for setting the COP problem, as we did not optimise that step). Moreover, the best result on each dataset (Figure 6.2) is rarely achieved with the latent representation with the most expressive Alp (with decoder clause length of 3, and any length of encoder clauses), which are the runs that take the longest. This suggest that the length of clauses serves as a form of regularisation, and one can safely stick to the shorter clauses (and stacking if the complexity is needed) which are fast to learn.

6.5 Conclusion

We introduce *Auto-encoding Logic Programs* (Alps) – a novel representation learning framework for representation learning with relational data. The novelty of the proposed framework is that it learns a latent representation in a symbolic, instead of gradient-based way. It achieves that by relying on first-order logic as a data representation language, which has a benefit of exactly representing the rich relational data without the need to approximate it in the embeddings spaces like many of the related works. We further show that learning Alps can be cast as a constraint optimisation problem, which can be solved efficiently in many cases. We experimentally evaluate our approach and show that learning from the relational latent representations created by Alps results in improved AUC-PR scores compared to learning in the original data representation.

Chapter 7

Towards a better understanding of relational latent representations

This chapter focuses on analysing the properties of relational latent representations. It focuses on explaining the effectiveness of such representation transformations and establishing the relative advantages of KGEs compared with the methods within the SRL methods and vice versa. This chapter is based on the following publications:

> DUMANČIĆ, S., AND BLOCKEEL, H. Demystifying relational latent representations. In *Inductive Logic Programming* (2018), N. Lachiche and C. Vrain, Eds., Springer International Publishing, pp. 63–77. Best Student Paper Award
>
> DUMANČIĆ, S., GARCIA-DURAN, A., AND NIEPERT, M. On embeddings as alternative paradigm for relational learning. In *Proceedings of the 8th International Worshop on Statistical Relational Artificial intelligence* (2018)
>
> DUMANČIĆ, S., GARCIA-DURAN, A., AND NIEPERT, M. A comparative study of distributional and symbolic paradigms for relational learning. *preparation* (2018)

7.1 Introduction

A common criticism of (relational) representation learning methods concerns their black-box nature. Latent features are created through layers of non-linear transformations which are often incomprehensible as the neural architectures implementing the transformation can easily have up to several billions of parameters. It is also difficult to know which kind of knowledge latent features capture and *why* that particular knowledge is useful for the prediction. As both entities and relations are mapped to the points in the Euclidean space, KGEs thus approximate the original data. We currently lack any measure to assess the quality of such approximation, nor do we know how to associate semantics to relations and entities in the Euclidean space.

This chapter introduces two sets of experiments that offer more insight into the effectiveness of relational latent representations. We first identify two properties that correlate well with the increase in the performance when the ILP model is learned from the latent representation created by CUR^{2}LED. We then compare several state of the art KGE and ILP methods on several benchmark datasets, including both relational classification and knowledge base completion. The experiments are designed to gain insights into the relative strengths and weaknesses of the respective methods. The experiments reveal several indicators that might help in choosing one method over another one for a specific relational dataset.

7.2 Why are CUR^{2}LED representations effective?

Latent features produced by CUR^{2}LED have proven useful in reducing the complexity of models and improving their performance. However, no explanation was offered why that is the case. In this section, we look into the properties of these latent representations and offer a partial explanation for their usefulness. To answer this question we introduce the following properties: label entropy, sparsity and redundancy.

Label entropy and sparsity serve as a proxy to a quantification of learning difficulty – i.e., how difficult is it to learn a definition of the target concept. Considering a particular predicate, label entropy reflects a *purity* of its true groundings with respect to the provided labels. Intuitively, if true groundings of predicates tend to predominantly focus on one particular label, we expect model learning to be easier.

Sparse representations, one of the cornerstones of deep learning [6], refer to a notion in which concepts are explained based on local (instead of global) properties of the instance space. Even though many properties might exist for a particular problem, sparse representations describe instances using only a small subset of those properties. Intuitively, a concept spread across a small number of local regions is expected to be easier to capture than a concept spread globally over an entire instance space.

Quantifying sparsity in relational data is a challenging task which can be approached from multiple directions – either by analysing the number of true groundings or interaction between entities, for instance. We adopt a simple definition: the number of true groundings of a predicate.

Label entropy and sparsity jointly describe a compelling property of data representation – instances space is divided into many local regions that match labels well and consequently make learning substantially easier.

Redundancy. A downside of CUR²LED is the high number of created features. Despite their proven usefulness, a high number of latent features enlarges the search space of a relational model and increases the difficulty of learning. As similarity interpretations are provided by the user, it is possible that almost identical clusterings are obtained with different similarity interpretations. Thus, if many of the features are redundant, removing them simplifies learning.

We measure the redundancy with the *adjusted Rand index* (ARI) [115, 94], a standard measure for overlap between clusterings, and study its impact on the performance. To evaluate the influence of redundant features, we modify CUR²LED by adding an additional *overlap parameter* α. Every time a new clustering is obtained, we check its overlap with the previously discovered clusterings using the ARI. If the calculated value is bigger than α, the clustering is rejected.

7.2.1 Experiments and results

We devise the experiments to answer the following questions:

(Q1) *Do latent features that result in models of lower complexity and/or improved performance exhibit a lower label entropy compared to the original data representation?*

(Q2) *Are latent representation that improve the performance of a model sparser than the original data representations?*

(Q3) *To which extent are latent features redundant?*

Datasets and setup

The results obtained in [35] can be divided in three categories. The first category contains the IMDB and UWCSE datasets; these datasets present easy relational learning tasks in which the original data representation is sufficient for almost perfect performance. The main benefit of latent representations for these tasks was the reduction of model complexity. The second category includes the TerroristAttack dataset, in which the main benefit of latent representation was the reduction of complexity, but not the performance. The third category involves the Hepatitis, Mutagenesis and WebKB datasets. These tasks benefited from latent representations in both performance and reduced model complexity. That is especially true for the Hepatitis and WebKB datasets on which the performance was improved by a large margin.

We take a representative task from each of the categories. Precisely, we use IMDB, UWCSE, Hepatitis and TerroristAttack datasets in our experiments. Both IMDB and UWCSE datasets were included as they are easy to understand without the domain knowledge, and thus useful for analysing the interpretability of relational latent features. As for the parameters of latent representation, we take the best parameters on individual datasets selected by the model selection procedure in [35]. When analysing the interpretability, we set θ to 0.3.

When evaluating the redundancy, we create latent representations by setting the α to the following values: $\{0.9, 0.8, 0.7, 0.6, 0.5\}$. We then learn a relational decision tree TILDE on the obtained representation and compare accuracies, the number of created features and the number of facts.

When analysing the entropy and sparsity of representations, predicates indicating labels (such as `Professor` or `Student`) and entity definitions (such as `Person` or `Course`) are not considered in the analysis.

Results

Label entropy. Figure 7.1 summarises the label entropy for each dataset. In all cases where representation learning proved helpful (i.e., IMDB, UWCSE, Hepatitis), latent representations have a substantially larger number of predicates with low label entropy compared to the original data representation. This means that the true groundings of latent predicate contain entities with the same label in a higher proportion than the predicate in the original data representation. The latent representation for the TerroristAttack datasets, however, shows a different behaviour in which latent features with high

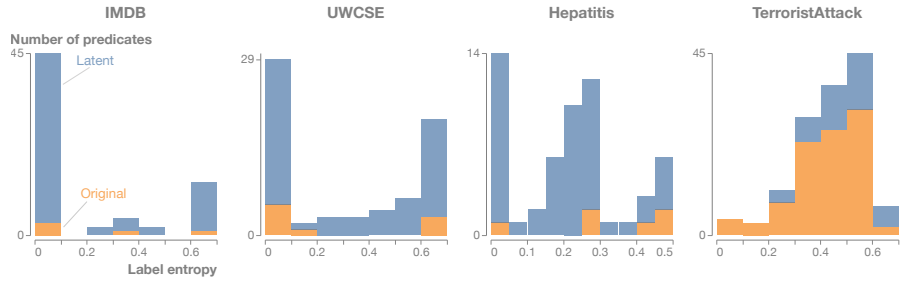


Figure 7.1: Latent representations for IMDB, UWCSE and Hepatitis datasets contain substantially larger number of predicates (and the corresponding facts) with low label entropy, compared to the original representation of data. On the TerroristAttack dataset, for which the latent representation has not been useful, that is not the case - both original and latent representation demonstrate similar trends in label entropy of the predicates and the corresponding facts.

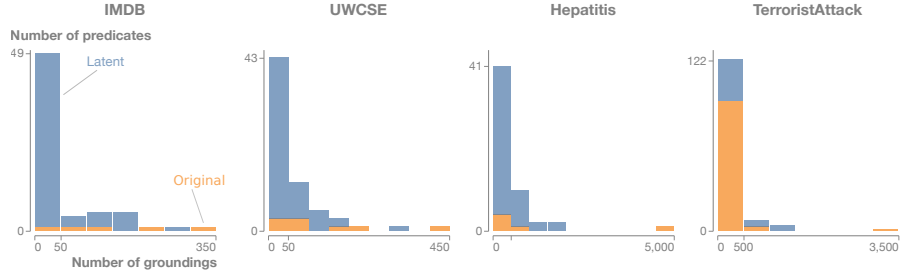


Figure 7.2: Latent representation tends to be sparser than the original representation on the datasets where it is beneficial (IMDB, UWCSE and Hepatitis). On the TerroristAttack dataset, where the latent representation is not beneficial, both original and latent representation follow the same trend.

entropy dominate the representation. These results agree with the expectation that a high number of low entropy features makes learning easier. However, not all latent features have low label entropy. This is expected, as the labels are not considered during learning of latent features. It also does not pose a problem – these latent features are less consistent with the one particular task, but it might well be the case that those features are useful for a different task.

Sparsity. Figure 7.2 summarises the sparsity results in terms of the number of true instantiations of predicates. The distribution of the number of true groundings in the latent representations (where latent features are beneficial)

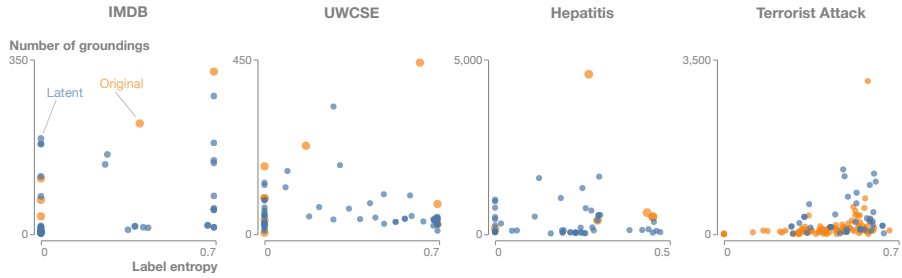


Figure 7.3: Contrasting the label entropy of predicates and the number of true groundings reveals that the many latent predicates with the low label entropy have similar number of groundings as the predicates of the original data representation. This means that the trivial case, in which a large number of low-entropy predicates is obtained due to many predicates that have just a few true groundings, is not explanation for the experimental results. Instead, the latent representation, when beneficial, successfully identifies local regions in the instance space that match well with the provided labels. The exception to this is again the TerroristAttack dataset.

is heavily skewed towards a small number of groundings, in contrast with the original representation. That is especially the case with the Hepatitis dataset, which profits the most from the latent features. The exception to this behaviour is again the TerroristAttack dataset in which the original representation already is very sparse. These results indicates that latent features indeed describe smaller groups of instances and their local properties, instead of global properties of all instances.

Connecting label entropy and sparsity. A potential explanation of the above discussed results might be that many latent features capture a very small number of instances (e.g., 1 or 2) which would lead to a large number of features with low label entropy. Such features would largely be useless as they make generalisation very difficult. To verify that this is not the case, Figure 7.3 plots the label entropy versus the number of groundings of a predicate. If latent features of low label entropy would indeed capture only a small number of instances, many points would be condensed in the bottom left corner of the plot. However, that is not the case – many latent predicates with low label entropy actually have a number of groundings comparable to the predicates in the original representation. The exception to this is again the TerroristAttacks dataset.

These results jointly point to the following conclusion: *latent features successfully*

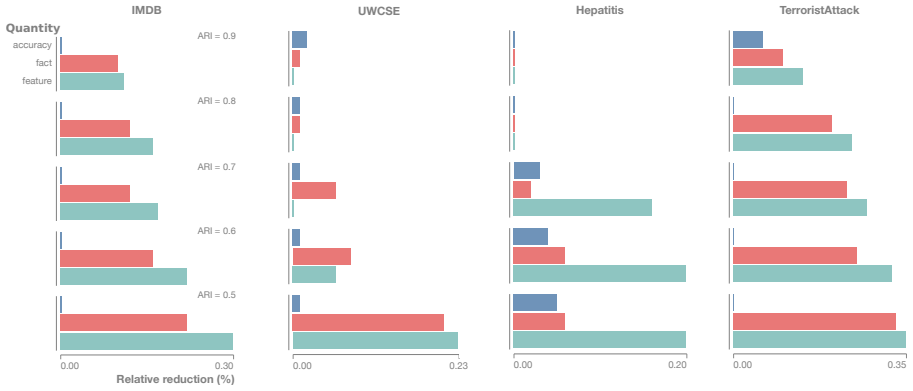


Figure 7.4: The performance in terms of the accuracy is barely effected by removing overlapping clusterings, while the number of predicates and facts can be reduced up to 30%. The only noticeable reduction in performance happen on the Hepatitis dataset, but only for approximately 5%.

identify local regions in the instance space that match well with the provided labels. As a consequence, these local regions are easier to capture and represent.

Redundancy. Figure 7.4 summarises the influence of α on the accuracy and the number of latent features. The figure shows relative reduction in the number of features (equal to the number of predicates), the number of facts and the accuracy with respect to the latent representation obtained without rejecting the overlapping clusterings. These results show that the performance of the classifier is not affected by removing features based on the overlap of clusterings they define. The performance of TILDE remains approximately the same, whereas the number of latent features is reduced by 20 to 30 %. As the number of features is directly related to the size of the search space of relational model (and thus the complexity of learning), this is an encouraging result indicating that the size of the search space can be naively reduced without sacrificing the performance.

7.3 On embeddings as an alternative paradigm for relational learning

SRL and KGE methods have largely been developed in isolation and little understanding is currently available on the relative advantages of the respective

approaches. The major strength of KGEs is their scalability – they easily operate on knowledge graphs with millions of facts and thousands of relations and the similarity of symbols emerging from the Euclidean space. As they essentially *vectorize* the data, the scalability of glskges benefits from very fast computational frameworks leveraging the strengths on GPUs. Symbolic SRL methods, on the other hand, are capable of capturing very complex relational patterns, are interpretable and flexible reasoning systems – once the model of the domain is obtained, a user can pose any query w.r.t. the model and does not have to commit to a predefined target.

The major weakness of KGEs is that vectorisation of complex relational data is necessarily an *approximation* of it, not the exact re-representation. We currently lack any measure of the quality of such a mapping. Furthermore, KGEs are black-box and uninterpretable, limited reasoning capabilities focused on local information, and have a difficult time handling unseen instances. The weakness of the symbolic SRL methods is that both inference and learning with such methods is highly complex, limiting its applicability, as well as lack of ways to represent similarity between symbols.

The two respective branches also focus on different tasks. KGE methods typically focus on knowledge graph completion which requires very simple forms of relational reasoning. The evaluation metrics are often measuring the quality of rankings generated by the respective scoring functions. Symbolic SRL methods typically focus on learning from small relational data, employing more complex forms of logical reasoning.

This work contributes towards a better understanding of the relative strengths and weaknesses of the aforementioned paradigms. We perform a systematic comparison of KGEs and symbolic SRL approaches on various standard relational classification, as well as the knowledge base completion tasks. Standard relational classification datasets offer a variety of tasks requiring various levels of reasoning complexity, while standard KBC datasets offer insights on real-life large knowledge graphs. We include both quantitative, in terms of performance, and qualitative analysis, in terms of extracted patterns of reasoning, in order to gain more insights into the suitability of these different methods.

7.3.1 Comparing symbolic and distributional methods

As stated before, the current understanding of the relative strengths and weaknesses of KGEs and symbolic SRL methods in very limited. Several works, however, do offer interesting insights. Nickel et al. ([102]) and Toutanova

and Chen ([139]) show that including both latent features from KGEs and the observable features, in the form of random walks over knowledge graphs, in a joint model can greatly increase the performance and reduce the learning complexity. Pujara et al. ([113]) show that KGEs have difficulties handling data with high degree of sparsity and noise – which is the case with every automatically created knowledge graph. Grefenstette ([57]) introduces a formula framework for simulating logical reasoning through tensor calculation, which can be seen as a form of embeddings that does not require learning.

The work most related to ours is that of Toutanova and Chen ([139]) and Vig et al. ([150]). Toutanova and Chen compare KGEs with a simple method that takes the types of incoming edges in a knowledge graph as features (termed observable patterns) and show that such simple observable patterns outperform KGEs on some datasets. However, they offer no greater insight in possible reasons. Vig et al. compare symbolic SRL methods with embeddings obtained by the Siamese neural network [15], and focus on analysing the impact of the available *background knowledge* on the performance. Their results indicate that KGEs might be beneficial when the background knowledge about the task at hand is limited but is such knowledge is available then the symbolic methods are preferable. Our work presented differs in a way that it goes beyond quantitative analysis and includes substantial qualitative analysis with respect to the complexity of reasoning needed to address the task.

7.3.2 Aims, Materials and Methods

The main goal of this study is to identify the strengths and weaknesses of KGE and symbolic SRL approaches to learning and reasoning with relational data. Concretely, we focus on the following question:

> *Are KGEs a viable alternative to symbolic methods for relational classification?*

More specifically, we focus on the following task

> *Given a set of **target instances** (entities in a knowledge base), learn a model that predicts the value of the labels associated with those instances. We consider fully relational models learning the logical theory for predicting the labels, and a feature based models learning from the vector representation of target instances.*

Our focus is exclusively on classification tasks as they give us a well-defined and clear performance measure, in contrast to the clustering task which is ill-defined [45].

KGE methods are usually trained to assign a high score to all facts in a knowledge base, not to specific *target* facts. To make sure this training criterion does not put KGEs at a disadvantage, once the embeddings are obtained, we use them as the input data for a classifier learning the predictive model for the pre-specified target.

As KGE methods are focused on the knowledge graph completion task, they assume that all entities are given in advance and focus on filling in the missing links in data. Therefore, they have difficulties with unseen instances. We do not address this issue here, but simply learn the representation of both training and test data at the same time (with labels excluded). It is, however, worth noting that KGE methods have a certain advantage due to this.

Materials

Relational classification dataset We take standard relational learning datasets and use the available labels as ground truth for both classification and clustering. The IMDB dataset is a snapshot of the Internet Movie Database, describing a set of movies with their actors and directors. The task is to distinguish between actors and directors. The UWCSE dataset describes the employees of the University of Washington, their roles, publications, and courses. The task is to distinguish between students and professors. The Mutagenesis dataset consists of a set of molecules and their structures, with the goal of predicting whether a compound is mutagenic or not. The Carcinogenesis data is a similar dataset, but focuses on a different compound property. The Yeast dataset describes a data about regulatory paths in the genome of yeast, and the target is to predict active paths. The WebKB dataset describes web pages of four US universities and their corresponding link structure. The pages are classified into seven groups according to their roles such as personal, departmental or project page. The Terrorists dataset contains a network of terrorist attacks, each assigned to of the 6 types of attacks. Finally, the Hepatitis dataset describes a set of patients with a diagnosis of Hepatitis B and C. We intentionally focus on standard relational learning datasets because they typically expose a more variety of reasoning: required reasoning ranges from attribute-only reasoning to multi-hop reasoning. Some of the properties are listed in Table 7.1. For each property, we differentiate between target instances – instances which contain labels, and the rest. This gives us a better opportunity to detect the conditions under which either of

Table 7.1: Dataset properties summarise the number of target and all instances, the number of attributes, and the number of relation types.

Dataset	Instances		Attributes		Relations	
	Target	Total	Target	Total	Target	Total
Hepa	500	5919	2	12	3	3
Muta	230	6124	3	7	3	7
Terror	1293	1293	104	104	2	2
WebKB	920	3880	763	1207	4	5
Carc	1700	45940	0	71	28	28
Yeast	13750	13750	1836	1836	1	1
UWCSE	2824	3714	0	23	1	5

the respective approaches is preferable. In contrast, standard KB completion datasets are often very simple and only simple path-type rules are required to achieve a reasonable performance. Therefore, the disadvantage of standard relational methods is that they might very inefficient because of the dataset size. Additionally, if the provided data does not contain useful features, KGE methods might still be able to extract certain useful patterns through the latent features.

Knowledge graph embeddings Due to the sheer magnitude of the existing KGE methods [153], including a large sample in the comparison would be infeasible. Moreover, it is not clear whether a big difference in performance is expected [67]. Therefore, we focus on the two prototypical and, arguably, most used approaches – TransE and DistMult, as well as the state-of-the-art approach of ComplEx [140]. It is important to note that ComplEx produces embeddings in the *complex Euclidean space* and that each entity is associated with two embeddings - *real* and *imaginary* one. In order to create a single embedding out of these two, we consider three version of ComplEx embeddings: (1) taking an average of two embeddings, (2) their sum, and (3) the concatenation of the two embeddings.

Classifiers When using machine learning algorithms for prediction, it is important to consider the bias an algorithm introduces. To better understand how the difference in performance between KGEs and symbolic SRL methods is influenced by the bias of the particular machine learning algorithm, we experiment with three different ML families – decision trees (DT), support vector machines (SVM) and k nearest neighbours (kNN), and their relational

counterparts – relational decision tree TILDE [13], relational kernel machines kFOIL [82] and kNN with a relational similarity measure of ReCeNT [36].

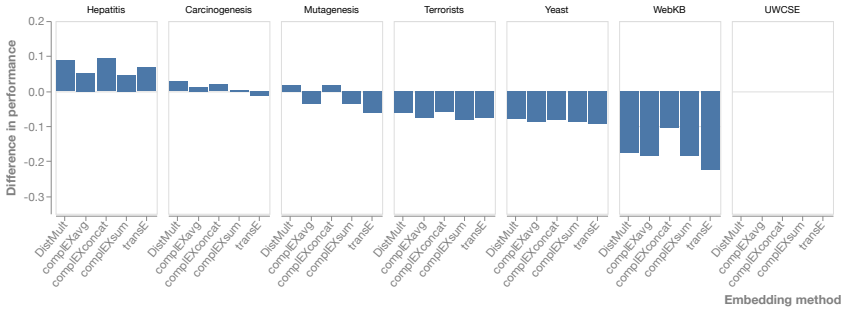
Methods

Relational classification We perform standard nested cross-validation. For each split, the training data is used to learn the models and tune their parameters (using an inner cross-validation loop) and the unseen fold is used for testing. We report the relative performance of KGEs to the relational baseline in terms of the difference in the accuracies, $acc_{KGE} - acc_{relational\ baseline}$, averaged over individual splits. The labels were excluded from the data when learning the KGEs and considered only during the training of the classifier.

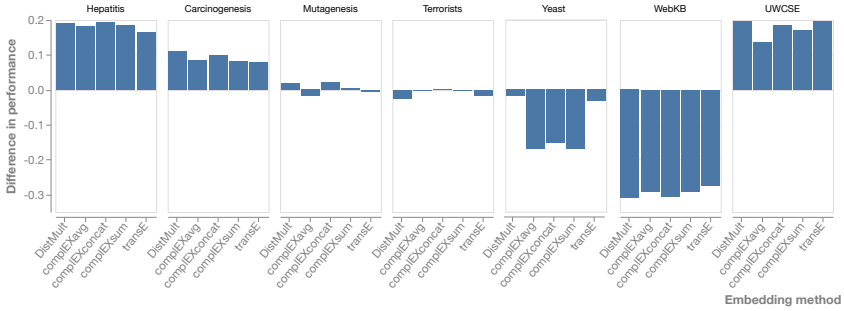
The embeddings of each relational data were obtained before learning a classifier. The dimensions of the embeddings were varied in $\{10, 20, 30, 50, 80, 100\}$; we include smaller dimensions because standard relational datasets tend to have a much smaller number of entities than the KBC datasets (see Table 7.1). All embeddings were trained to 100 epochs and saved in steps of 20. We do not use the validation set and metrics such as mean reciprocal rank to select the best hyper-parameters for the embeddings, as they may not be perfectly correlated with the predictive accuracy; instead, we treat the dimension and the number of epochs for training as additional parameters while training the classifiers as part of the inner cross-validation loop.

Knowledge base completion For the experiments with the knowledge base completion datasets, we do not re-train the KGEs but report the results from [31] which are considered to be state-of-the-art. These experiments include several methods: DistMult [159], ComplEx [140], R-CGN [126] and ConvE [31]. Relational decision tree TILDE is used as a relational baseline due to its simplicity and speed. One model is trained for each relation, and the mean and weighted (by the number of examples in the relation) accuracy is reported. The training procedure is the same as for the experiments regarding the relational classification datasets, with an exception that only one (pre-defined) test split is used. KGEs are usually evaluated by how well they rank the correct answers to a triple in which either the subject or objects are left out (i.e, $< \mathbf{s}, \mathbf{r}, ? >$ or $< ?, \mathbf{r}, ? >$), and the metrics used are ranking measure such as *mean reciprocal rank* and *hits @ K* (*is the correct answer among the top K ranked answers*) [14]. Such rankings are difficult to obtain with the ILP methods, which makes the comparison somewhat difficult. As Hits@1 directly correspond to the accuracy, and we use this as a primary mean of the comparison.

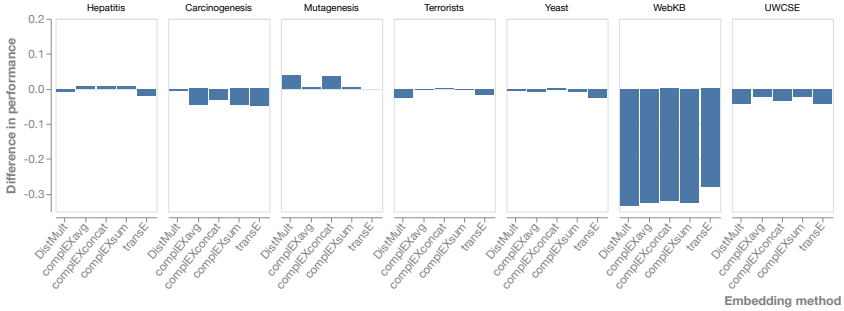
7.3.3 Comparative results



(a) Performance with decision trees



(b) Performance with svms



(c) Performance with kNNs

Figure 7.5: Difference in performance between sub-symbolic and symbolic methods. The values *above* the line indicate that sub-symbolic methods perform better, while the values below the line indicate the cases on which symbolic methods perform better.

The experiments with (relational) decision trees (Figure 7.5a) indicate that the decision tree trained on the embeddings features outperform TILDE only on two datasets– Hepatitis and Carcinogenesis, while performance is equal on the UWCSE dataset. The results also indicate that the performance of various KGE models is relatively similar, but *DistMult* and *ComplEXconcat* seem to have a slight edge.

What is most interesting in these results is not the difference in the performance itself, but *whether we could explain what gives KGEs an advantage on certain datasets and disadvantage on others?* In order to do so, we analyse the results with respect to the semantics of the predicates used in the extracted rules, which yields further interesting results. More precisely, we differentiate between *attribute predicates* that associate a specific value of an attribute with an instance, and *relation predicates* that connect two instances. We further distinguish between *target* instances having an associated label, and *neighbourhood* instances that provide additional data. Table 7.2 summarises the rules extracted by TILDE w.r.t. the above-discussed interpretation.

Table 7.2: Properties of extracted rules are summarised as a proportion of relation and attribute predicates they contain. *Rule properties* section describes the proportion of rules that contain *relation* predicates only, *attribute* predicates only and the mix of both, for each dataset. *Dataset properties* section shows the proportion of possible attribute and relation predicates that are used in the extracted rules. For example, the rules on the Hepatitis dataset use 100 % of the possible attribute predicates, and 66 % of relation predicates. *Neighbourhood* refers to the proportion of attribute and relation predicates of the non-target instances for which a direct link with target instances exists.

Dataset	**Model**	**Rule properties**			**Dataset properties**		
	complexity	Relations	Attributes	Mix	Attributes	Relations	Neighbourhood
Hepatitis	15	10 %	38 %	52 %	100 %	66 %	70 %
Carcinogenesis	24	0 %	0 %	100%	–	100 %	72 %
UWCSE	35.5	0 %	2 %	98 %	75 %	80 %	0 %
Terrorists	17.65	0 %	95 %	5 %	35 %	100 %	0 %
Mutagenesis	9.4	0 %	100 %	0 %	100 %	0 %	0 %
WebKB	38	5 %	76 %	19 %	4 %	50 %	1 %
Yeast	188.4	0 %	8 %	92 %	78 %	100 %	–

The results indicate an interesting connection between the performance and the amount of available information that is found predictive: the properties of the TILDE rules extracted for the Hepatitis and Carcinogenesis datasets reveal that the majority of provided information, both attributes and relations, are found to be predictive. We define the predictive information as follows: the proportion of available attribute and relationship information that a learned symbolic approach uses, i.e., the proportion of available attributes and relationships that

are present in the extracted rules. For instance, on the Hepatitis dataset, the induced rules incorporate 100 % of the target attributes and 66 % of relations involving target instances, as well as 70 % of the predicates related to the neighbouring instances. Similar numbers hold for the Carcinogenesis data.

The remaining datasets are different in that regard – only a fraction of information is found predictive. On the Mutagenesis dataset[1], even though all the attributes of molecules are found predictive, none of the relations were found predictive and thus none of the information about atoms they contain. Thus, only a fraction of available information is used. On the WebKB dataset, only 4 % of the attributes have been found predictive, as well as 50 % of relations.

The results and the analysis of the rules suggest that KGEs have an edge over the SRL methods when *most of the information in the neighbourhood of instances is relevant*. An explanation for such a big difference in performance might be that SRL methods have to select few predictive rules while learning theories, and thus can discard irrelevant information. In contrast, the learning principle of KGEs is designed to take all of the information into account, which seems to make them underperform.

A similar trend is also evident when the (relational) SVM is used as a classifier. However, KGE-based methods achieve better relative performance on the Hepatitis, Carcinogenesis and UWCSE dataset, while the difference in performance is much less pronounced on the Mutagenesis and Terrorists datasets. It is interesting to note that symbolic and KGE-based methods do not benefit equally by using the more powerful learner. We attribute this to the fact that kFOIL essentially creates a *binary embedding* – it identifies a set of useful relational features (similar to random walks), followed by learning a kernel SVM on such binary data. Therefore, it is closer to the KGE-based methods than the fully relational once, and it is likely that it loses some of the semantics of relational data by binarising it.

The pattern, however, disappears with the kNN classifier. Whereas previously KGE-based methods substantially outperformed symbolic methods on the Hepatitis, Carcinogenesis and UWCSE datasets, relational versions of kNN perform equally well on the Hepatitis dataset and outperform KGE-based methods on the Carcinogenesis and UWCSE datasets. KGE-based methods seem to gain the advantage on the Mutagenesis dataset, but the remaining datasets prefer symbolic approaches. However, a general trend is that the differences in performances are much less pronounced compared to the results with decision trees and SVMs, except on the WebKB. We suspect that the reason

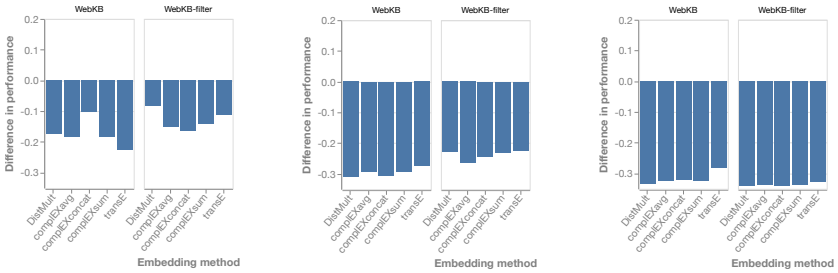
[1] we use the version of the dataset without any rings structures and additional background knowledge which is difficult to incorporate in KGE methods

for this is that similarity is difficult to measure with relational data, despite a considerable body of work addressing the problem [36], and KGEs might be a viable alternative.

To validate our conclusion, we design a control experiment focusing on the WebKB dataset where the difference in performance is the largest. We created the second version of the dataset that contains only the information, both relations and attributes, deemed useful by the symbolic SRL methods. More precisely, in the case of decision trees, we take the information deemed useful by decision trees, while in the case of SVM we take the information found useful by kFOIL. For the case of kNN, it is not easy to identify the exact information that makes the most influence on the relational similarity measure; therefore, we use the information identified by TILDE and kFOIL. If our observation is correct and not the result of randomness, the performance of KGEs should improve on this filtered version of the dataset. The results (Figure 7.6) do confirm so: the difference in performance of KGEs and symbolic SRL methods reduces in the cases of decision trees (Figure 7.6a) and SVMs (Figure 7.6b), especially when decision trees are used. Interestingly, TransE and DistMult seem to benefit from that more than ComplEx. The differences in performance between KGEs and symbolic SRL methods do not disappear completely – we attribute this to the sparsity of the filtered graph, as it is known that the quality of the embeddings relies on the level sparsity of the knowledge graph [113]. Moreover, filtering the data might have filtered out some of the useful information that coincidentally correlates well with information deemed useful by the symbolic learner. Thus, the inclusion of such information might be beneficial to KGEs, but we do not have a way to detect it easily. Moreover, we have noticed that there is a relatively small overlap between information found predictive by TILDE and kFOIL, indicating that the bias of the learning algorithm plays an important role in detecting which parts of the available information is predictive. The kNN results again diverge from this trend, and the results suggest that KGE-based methods perform worse than with the unfiltered dataset. The likely reason for that is that the filtered information matches the bias of decision trees and SVMs, but not the kNNs similarity measure, and further confirms that measuring the similarity in relational data is a difficult problem.

7.3.4 Key messages

Many problems nowadays are naturally expressed in form of relational and graph structures data. This includes social and protein interaction networks, biological data, knowledge graphs and many more. Two main machine learning paradigms for analysing such data, knowledge graphs embeddings and symbolic statistical relational learning, have mostly been studied in



(a) Decision trees (b) SVM (c) kNN

Figure 7.6: Results on the filtered version of the WebKB dataset show the improvement in the performance of KGE-based methods in combination with decision trees and SVMs, but not when kNN is used.

isolation so far. This work is the first, to the best of our knowledge, that systematically compares these two paradigms on the standard tasks from both domains – relational classification and knowledge base completion. Our results point to the following conclusion:

- **KGEs seems to be suitable for curated data.** KGEs seem to be at a disadvantage when only a small fraction of available information is useful for the given prediction task. Symbolic SRL methods do not have that issue as they can cherry pick useful information during the learning phase. This conclusion was further confirmed by the control experiment in which the uninformative data (as found by the symbolic methods) was removed from the dataset – the difference in performance between KGEs and symbolic methods has decreased.

- **Symbolic methods outperform KGEs on knowledge base completion tasks.** We have noticed a large gap in the performance between symbolic and KGE methods when compared on the standard KBC datasets: whereas symbolic methods achieve accuracy $> 90\%$, KGE methods struggle and achieve the accuracy $\approx 50\%$. This suggests that the KGEs, at least in the current state, are more suitable for the interactive inspections of large knowledge graphs than for the automatic reasoning.

- **KGEs might be a good way to measure the similarity in relational data.** On most datasets, KGE and symbolic methods had almost identical performance when learning based on similarities was employed. This suggests that KGE-based similarity is competitive with the large body of works on this problem.

This work is not meant as a criticism of any of the considered approaches but hopes to identify *useful bits* of individual approaches that might be worth integrating. We hope this work inspires new research directions focused on combining the strengths of both approaches as both communities has already started to explore [93, 30, 126].

Intermezzo: DeepProbLog

Symbolic methods for representation learning with relational data are in the focus of this thesis. The experimental analysis presented in this chapter clearly shows that both research directions have strengths and weaknesses. Moreover, a general consensus is that deep learning excels at perception tasks such as object recognition in images, whereas symbolic approaches excel at reasoning. Therefore, investigating the closer integration of symbolic and deep approaches deserves attention.

Manhaeve et al. [89] introduce `DeepProbLog` – an extension of `Problog` with the support for *neural predicates*. `DeepProbLog` starts from the perspective that explicitly specifying the probabilities of all probabilistic facts in a domain might be difficult for certain tasks, if not impossible. For instance, various perception tasks require large image datasets for learning a well-performing model; manually annotating the probabilities of certain objects being detected in all images is impossible. the core idea of `DeepProbLog` is to introduce a new construct, the *neural predicate*, that allows a user to provide a *neural model* that can estimate the probabilities of grounded atoms being `true` given the arguments of the atom. The neural model does not have to be trained beforehand, but can be learned from scratch given the examples of a target task.

Formally, `DeepProbLog` extends the concept of *annotated disjunction*. Annotated disjunction is an expression of the form

$$\mathtt{p}_1\mathtt{::h}_1\mathtt{;}\ \ldots\mathtt{;}\ \mathtt{p}_n\mathtt{::h}_n\ \mathtt{:-}\ \mathtt{b}_1\mathtt{,}\ldots\mathtt{,b}_m\mathtt{.}$$

where $\mathtt{p}_i$ are probabilities to that $\sum_i p_i = 1$, and $\mathtt{h}_i$ and $\mathtt{b}_j$ are atoms. The meaning of the annotated disjunction is that whenever all $\mathtt{b}_i$ are `true`, $\mathtt{h}_j$ will be true with probability $\mathtt{p}_j$ with all other $\mathtt{h}_{i\neq j}$ being `false`. `DeepProbLog` introduces a *neural* annotated disjunction of the form

$$nn(m_q, \vec{t}, \vec{u}) :: q(\vec{t}, u_1); \ldots; nn(m_q, \vec{t}, \vec{u}) :: q(\vec{t}, u_n)\ \mathtt{:-}\ \mathtt{b}_1\mathtt{,}\ldots\mathtt{,b}_m\mathtt{.}$$

where nn is a neural model, $\vec{t}$ is a vector of ground terms representing the inputs of a neural model for predicate q, u_i are possible output values of the neural model. m_q is the identifier of a neural model that specifies a probability distribution over output values $\vec{u}$ given the output $\vec{t}$

7.4 Conclusion

This chapter digs deeper into the properties of relational latent representations. It first inspects latent representations created by CUR^{2}LED showing that such latent representations improve the performance of relational learners by capturing small local regions in data that match well with the labels. It demonstrates that by showing that the predicates in the latent representation are sparser but with the reduced label entropy. The analysis also shows that the CUR^{2}LED-created latent features tend to be redundant and shows that the number of latent features can be reduced without sacrificing the performance. The chapter then focuses on comparing the KGE and ILP approaches to relational learning on various benchmarks. This second analysis shows that KGEs outperform ILP methods on relational classification tasks when the data is curated – the data contains only the relevant information. In the scenarios when that is not the case, ILP have an edge over the KGE methods. The analysis also shows that ILP methods outperform KGEs on the standard knowledge base completion tasks, while embeddings seem to be an interesting approach for assessing the similarity of relational objects.

Chapter 8

Conclusion

We conclude by summarising the presented work, restating its main contributions, and providing an outlook on future research.

8.1 Thesis summary and contributions

The success of any machine learning application depends on the quality of the provided data. It reflects the quality of the data collection process and data representation. The quality of data representation is reflected in the quality of features that are used to describe the desired objects, which should not only the informative and relevant but should also allow capturing interesting patterns easily. This usually requires a domain expert knowing which features might be relevant for the task. What used to be a time-consuming and labour-intensive task is now transformed into an automatic procedure by *representation* or *deep learning*. The field of representation learning concerns the development of techniques that automatically extract an effective representation of the provided data and has revolutionised the field of machine learning over the last decade.

Unfortunately, the majority of the existing representation learning methods focuses on *flat data*, which represents data with vectors containing features values. This is in contrast with the real world which contains objects and their mutual relationships, i.e., the real world is *inherently relational*. The existing representation learning approaches addressing the relational data typically resort to *flattening* the relational data into vectors. This mapping can only *approximate* the relational data, and we currently lack a quantitative measure

to assess the quality of the approximation. Moreover, these approaches do not leverage the tools developed within *statistical relational learning* which concerns learning from relational data. Statistical relational learning leverages the expressive power of first-order logic to represent such complex data which makes it amongst the most powerful machine learning frameworks.

This thesis serves as an exploration of ideas that combine representation learning with statistical relational learning that, instead of resorting to data flattening, rely on first-order logic as a representation language for both data representation and latent feature representation. This thesis contributes several algorithms, techniques and analyses towards achieving this goal. We now summarise the main contributions of the thesis.

Expressive relational clustering framework The first contribution is a novel relational clustering framework. The main novelty of the proposed framework is a versatile dissimilarity measure that does not impose a fixed view on what makes relational objects similar, but is a *composition* of several *primitive* similarities. These primitive similarities include attribute-based similarity, proximity and various similarities of neighbourhoods. The experiments show that accounting for a diverse set of primitive similarities is beneficial for both relational clustering and classification, outperforming the approaches that consider only one (or a few) of the primitive similarities.

Exploiting symmetries to define relational latent features The second contribution is the technique for inventing relational latent representations by capturing approximate symmetries in data. The proposed technique relies on the previously introduced relational clustering framework to identify such symmetries by means of clustering. The main novelty this technique introduces is that it clusters objects and relationships in a data using different *similarity interpretations*, instead of composing several of them into one joint complex similarity measure. Compared to the existing work that uses clustering to enhance relational representations, the proposed technique clusters the instances of relationships, not just their types. It can thus discover various *sub-types* of relationships that might carry useful information. We show that learning from the latent representations created by clustering is beneficial for relational learning as it often improves the performance of the relational classifier while at the same time reducing the complexity of the induced model. Moreover, we introduce a relatively simple method for explaining the invented latent features which allow us to overcome the typical issue of a black-box feature invention machines.

These two contributions motive us to pose the following claim:

> **Claim.** To successfully address relational learning tasks, one has to account for several sources of relevant information.

The results related to the first two contributions show that the reason why relational learning is difficult is that relevant information comes from different sources, being it the attributes or surrounding structure and that separating those sources helps to improve the performance. This is related to the concept of *disentangling the factors of variation* many deep learning approaches aspire to achieve.

Auto-encoding logic programs The third contribution of this thesis are auto-encoding logic programs. They stand for a generalisation of the flat auto-encoder towards logic programs as a representation language. That includes both data and the computational framework for defining relational features. In contrast to CUR^{2}LED which introduces relational concepts into relational representation learning but still relies on statistical summaries for inventing the latent features, Auto-encoding logic program rely on clausal logic for every step of the pipeline. Therefore, they fully leverage the expressiveness of logic. We introduce a constraint optimisation framework for learning auto-encoding logic programs that proved to be reasonably efficient. The experiments show that latent representations created by Alp aid learning generative SRL models, resulting in a better performance of the SRL models learnt in the latent space.

Analysis of relational representation learning paradigms The fourth contribution of this thesis is an analysis of relational representation learning techniques. We start by analysing the relational latent representations created by CUR^{2}LED and show that such representations are effective because the identify groupings in data that correspond well with the available labels. We then focus on comparing the symbolic and embedding SRL approaches on a set of standard benchmarks for relational classification and knowledge base completion. The results of the comparison show that embedding approaches are suitable for scenarios when the data is *curated* and most of the available information is found predictive for the task at hand. Symbolic approach like SRL are more suitable for the case when data cannot be curated and reduced to only the relevant information. Finally, embeddings SRL approaches seem to be a good alternative for assessing the similarity in relational data.

In summary, we contribute novel methods for representation learning with relational data. The methods introduced in this thesis are *symbolic* in their nature, which is in stark contrast with the existing methods which rely on gradient-based optimisation. This allows them to leverage a powerful representation language of clausal logic to learn more expressive latent features. Our second main contribution is a better understanding of the strengths and weaknesses of symbolic and embeddings representation learning approaches.

8.2 Future work

We believe the methods introduced in this thesis clearly demonstrate the benefit of retaining logic as a representation language for relational representation learning. The most important benefit is the expressiveness of predicate logic that can represent very complicated interactions in data, various forms of reasoning while at the same time being interpretable. We hope these results will inspire others to pursue this direction of research. Here we discuss directions for future research, grouping them in the theoretical and practical aspects.

8.2.1 Theoretical aspects

Properties of good relational data representations *What constitutes a good data representation* is one of the pressing open questions in this line of research. Answering this question is the critical step that will shape the development of novel relational representation techniques. As we have outlined in Chapter 3, we do have a good intuition what we consider to be an effective *propositional* representation (Section 3.2.3 in Chapter 3): hierarchically organised, smooth, sparse and so on. Most of these properties are intuitions that come from empirical observation, not strong theoretical results. Moreover, this thesis has proven that mapping those concepts to relational data representations is not a straightforward task. Part of the reason is the semantic difference between Euclidean spaces and first-order logic. Understanding which data properties help relational learning will help us tune the latent representation created by CUR^{2}LED and Alps. This is especially important for Alps as it can inform us how to impose the language bias in order to discover the latent features.

Different representation learning techniques This thesis has explored two representation learning approaches – *learning by clustering* and the *auto-encoding principle*. The literature on representation learning consists of many different, and effective, ideas. It is well worth exploring various ideas, and one particularly interesting idea are *Generative adversarial networks* [56], which learns a generative latent model through a competition with a *discriminator* network trying to distinguish between the real data and the samples of the generative model. One of the main challenges here is how to propagate the feedback from the discriminator to the generator network, and the bi-directionality of Prolog predicates might be an elegant way to tackle this.

Do we need probabilistic latent representations? In this thesis, we have treated relational representation learning as a deterministic mapping from the original data space to the latent one. Thus, we have ignored the probabilistic aspect of SRL methods. An interesting question for the future research is whether it is necessary to include the probabilistic reasoning in the mapping from the original to the latent representation. It is certain that including the probabilistic component would impact the efficiency of learning, but we cannot state for sure that it would allow us to construct substantially better representations because the relational data is still deterministic. However, such probabilistic relational representation might be beneficial, and necessary, for the case of probabilistic data, e.g., often used in Problog [29].

Integrating symbolic and sub-symbolic representation learning methods This thesis considers learning relational latent representations from the perspective of logical approaches to SRL, and is focused on learning latent representation of symbolic data. Many existing approaches for representation learning considers numerical and signal-like data and effectively treat the symbolic data in the same way by re-representing it in the Euclidean space. This sacrifices many aspects and expressivity of logic. We believe the middle ground, which exploits the best parts of both logic and neural networks, is the direction that could show the most advancement. A good step in that direction is *DeepProbLog* [89], which elegantly integrates high level reasoning capabilities of logic with the perception achieved with ANNs. We believe another avenue worth exploring is a development of *hybrid* learning that would exploit strengths of both symbolic and distributional approaches for relational learning, similar in direction to differentiable theorem proving [121], but without sacrificing the expressivity of individual components.

8.2.2 Practical aspects

Scalable learning of relational representations This thesis considers laying down the foundations for a logic-based relational representation learning, and a lot of work still has to be done in that direction. But making these methods applicable to real-world scenarios, the scalability is an important issue to address. We do not pay special attention to this issue during the thesis. Improving the scalability of CUR^{2}LED involves investigating scalable clustering methods, as the calculation of the similarity measure is not the most computationally expensive step in the pipeline. Alps, on the other hand, can benefit from improving the constraint optimisation procedure by exploring

novel problem encodings, decomposition techniques [17, 50] and new semantic constraints.

Task-specific representations The goal of learning latent representations of data is currently focused on improving the performance of a predictive model, either through improving the predictive accuracy of the model or making the learning easier. This is the setup we have focused in this thesis. However, there is no reason to stop there: having a model of a domain, i.e., facts and clauses that are known to be true, SRL systems can reason about any aspect of the domain. We believe this open many doors for relational representation learning, where one interesting avenue might be to learn representations for faster inference and tackling the challenge of *lifted inference* [146, 111].

Learning how to *see* relational data Perhaps the most important idea from Chapters 4 and 5 is that explicitly accounting for multiple sources of information within relational data can substantially improve the performance. However, in both chapters we have relied on manually specifying these source; for clustering in Chapter 4 we have used all sources in a linear combination, while in Chapter 5 we have decided to use the attribute information, neighbourhood structures and the combination of both. An interesting issue to address is whether we could develop methods to identify these sources automatically. Our relational representation learning methods CUR^{2}LED would benefit substantially from this development as it could potentially discover more complex latent features without an intervention of the user.

Bibliography

[1] AHUJA, R. K., ERGUN, O., ORLIN, J. B., AND PUNNEN, A. P. A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math. 123*, 1-3 (Nov. 2002), 75–102.

[2] ARBELAITZ, O., GURRUTXAGA, I., MUGUERZA, J., PÉREZ, J. M., AND PERONA, I. N. An extensive comparative study of cluster validity indices. *Pattern Recogn. 46*, 1 (Jan. 2013), 243–256.

[3] BADER, D. A., MEYERHENKE, H., SANDERS, P., AND WAGNER, D., Eds. *Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings* (2013), vol. 588 of *Contemporary Mathematics*, American Mathematical Society.

[4] BAI, L., REN, P., AND HANCOCK, E. R. A hypergraph kernel from isomorphism tests. In *Proceedings of the 2014 International Conference on Pattern Recognition* (Washington, DC, USA, 2014), ICPR '14, IEEE Computer Society, pp. 3880–3885.

[5] BENGIO, Y. Learning deep architectures for ai. *Found. Trends Mach. Learn. 2*, 1 (Jan. 2009), 1–127.

[6] BENGIO, Y., COURVILLE, A., AND VINCENT, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell. 35*, 8 (Aug. 2013), 1798–1828.

[7] BENGIO, Y., AND DELALLEAU, O. On the expressive power of deep architectures. In *Proceedings of the 22Nd International Conference on Algorithmic Learning Theory* (Berlin, Heidelberg, 2011), ALT'11, Springer-Verlag, pp. 18–36.

[8] BENGIO, Y., LAMBLIN, P., POPOVICI, D., AND LAROCHELLE, H. Greedy layer-wise training of deep networks. In *NIPS* (Cambridge, MA, USA, 2006), NIPS'06, MIT Press, pp. 153–160.

[9] BENGIO, Y., LAMBLIN, P., POPOVICI, D., AND LAROCHELLE, H. Greedy layer-wise training of deep networks. In *NIPS* (2007), MIT Press.

[10] BICKEL, S., AND SCHEFFER, T. Multi-view clustering. In *Proceedings of the Fourth IEEE International Conference on Data Mining* (Washington, DC, USA, 2004), ICDM '04, IEEE Computer Society, pp. 19–26.

[11] BILLE, P. A survey on tree edit distance and related problems. vol. 337, Elsevier Science Publishers Ltd., pp. 217–239.

[12] BLOCKEEL, H. Bias specification language. In *Encyclopedia of Machine Learning and Data Mining*. Springer, 2017, pp. 125–128.

[13] BLOCKEEL, H., AND DE RAEDT, L. Top-down induction of first-order logical decision trees. *Artificial Intelligence 101*, 1–2 (1998), 285 – 297.

[14] BORDES, A., USUNIER, N., GARCIA-DURÁN, A., WESTON, J., AND YAKHNENKO, O. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems* (USA, 2013), NIPS'13, Curran Associates Inc., pp. 2787–2795.

[15] BROMLEY, J., GUYON, I., LECUN, Y., SÄCKINGER, E., AND SHAH, R. Signature verification using a "siamese" time delay neural network. In *NIPS* (San Francisco, CA, USA, 1993), NIPS'93, Morgan Kaufmann Publishers Inc., pp. 737–744.

[16] CAMACHO, R., FONSECA, N. A., ROCHA, R., AND COSTA, V. S. ILP : - just trie it. In *Inductive Logic Programming, 17th International Conference, ILP, Corvallis, OR, USA* (2007), pp. 78–87.

[17] CHAIEB, M., JEMAI, J., AND MELLOULI, K. A four-decomposition strategies for hierarchically modeling combinatorial optimization problems: framework, conditions and relations. In *2015 International Conference on High Performance Computing Simulation (HPCS)* (July 2015), pp. 491–498.

[18] COATES, A., LEE, H., AND NG, A. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011), vol. 15 of *JMLR Workshop and Conference Proceedings*, JMLR W&CP, pp. 215–223.

[19] COOK, D. J., AND HOLDER, L. B. *Mining Graph Data*. John Wiley & Sons, 2006.

[20] CRAVEN, M., AND SLATTERY, S. Relational learning with statistical predicate invention: Better models for hypertext. *Mach. Learn. 43*, 1-2 (Apr. 2001), 97–119.

[21] CROPPER, A., AND MUGGLETON, S. Can predicate invention compensate for incomplete background knowledge? IOS Press, pp. 27–36.

[22] CROPPER, A., AND MUGGLETON, S. Learning efficient logical robot strategies involving composable objects. In *IJCAI* (2015).

[23] CROPPER, A., AND MUGGLETON, S. H. Learning efficient logic programs. *Machine Learning* (Apr 2018).

[24] CROPPER, A., TAMADDONI-NEZHAD, A., AND MUGGLETON, S. H. Meta-interpretive learning of data transformation programs. In *ILP* (2015), vol. 9575 of *Lecture Notes in Computer Science*, Springer, pp. 46–59.

[25] DAVIS, J., AND GOADRICH, M. The relationship between precision-recall and roc curves. In *ICML* (New York, NY, USA, 2006), ICML '06, ACM, pp. 233–240.

[26] DAVIS, J., ONG, I. M., STRUYF, J., BURNSIDE, E. S., PAGE, D., AND COSTA, V. S. Change of representation for statistical relational learning. In *IJCAI* (2007), M. M. Veloso, Ed., pp. 2719–2726.

[27] DE RAEDT, L. Attribute-value learning versus inductive logic programming: The missing links. In *Inductive Logic Programming* (Berlin, Heidelberg, 1998), D. Page, Ed., Springer Berlin Heidelberg, pp. 1–8.

[28] DE RAEDT, L. *Logical and relational learning*. Cognitive Technologies. Springer, 2008.

[29] DE RAEDT, L., KIMMIG, A., AND TOIVONEN, H. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (San Francisco, CA, USA, 2007), IJCAI'07, Morgan Kaufmann Publishers Inc., pp. 2468–2473.

[30] DEMEESTER, T., ROCKTÄSCHEL, T., AND RIEDEL, S. Lifted rule injection for relation embeddings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016* (2016), pp. 1389–1399.

[31] DETTMERS, T., PASQUALE, M., PONTUS, S., AND RIEDEL, S. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence* (February 2018).

[32] DIETTERICH, T. G., LATHROP, R. H., AND LOZANO-PÉREZ, T. Solving the multiple instance problem with axis-parallel rectangles. *Artif. Intell. 89*, 1-2 (Jan. 1997), 31–71.

[33] DUMANČIĆ, S., AND BLOCKEEL, H. An efficient and expressive similarity measure for relational clustering using neighbourhood trees. In *ECAI*, vol. 285 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 1674–1675.

[34] DUMANČIĆ, S., AND BLOCKEEL, H. Unsupervised relational representation learning via clustering: Preliminary results. In *6th workshop on Statistical Relational Artificial Intelligence StarAI at IJCAI* (2016).

[35] DUMANČIĆ, S., AND BLOCKEEL, H. Clustering-based relational unsupervised representation learning with an explicit distributed representation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17* (2017), pp. 1631–1637.

[36] DUMANČIĆ, S., AND BLOCKEEL, H. An expressive dissimilarity measure for relational clustering using neighbourhood trees. *Machine Learning 106*, 9-10 (Oct. 2017), 1523–1545.

[37] DUMANČIĆ, S., AND BLOCKEEL, H. Demystifying relational latent representations. In *Inductive Logic Programming* (2018), N. Lachiche and C. Vrain, Eds., Springer International Publishing, pp. 63–77. Best Student Paper Award.

[38] DUMANČIĆ, S., GARCIA-DURAN, A., AND NIEPERT, M. A comparative study of distributional and symbolic paradigms for relational learning. *preparation* (2018).

[39] DUMANČIĆ, S., GARCIA-DURAN, A., AND NIEPERT, M. On embeddings as alternative paradigm for relational learning. In *Proceedings of the 8th International Worshop on Statistical Relational Artificial intelligence* (2018).

[40] DUMANČIĆ, S., GUNS, T., MEERT, W., AND BLOCKEEL, H. Auto-encoding logic programs. In *Proceedings of the 2nd Workshop on Neural Abstract Machines and Program Induction* (2018).

[41] DUMANČIĆ, S., GUNS, T., MEERT, W., AND BLOCKEEL, H. Auto-encoding logic programs. *preparations* (2018).

[42] DUMANČIĆ, S., MEERT, W., AND BLOCKEEL, H. Theory reconstruction: a representation learning view on predicate invention. In *6th workshop on Statistical Relational Artificial Intelligence StarAI at IJCAI* (2016).

[43] DZEROSKI, S., AND BLOCKEEL, H. Multi-relational data mining 2004: workshop report. *SIGKDD Explorations 6*, 2 (2004), 140–141.

[44] EMDE, W., AND WETTSCHERECK, D. Relational instance based learning. In *Proceedings 13th International Conference on Machine Learning (ICML 1996), July 3-6, 1996, Bari, Italy* (1996), L. Saitta, Ed., Morgan-Kaufman Publishers, San Francisco, CA, USA, pp. 122–130.

[45] ESTIVILL-CASTRO, V. Why so many clustering algorithms: A position paper. *SIGKDD Explor. Newsl. 4*, 1 (June 2002), 65–75.

[46] FERRUCCI, D. A., BROWN, E. W., CHU-CARROLL, J., FAN, J., GONDEK, D., KALYANPUR, A., LALLY, A., MURDOCK, J. W., NYBERG, E., PRAGER, J. M., SCHLAEFER, N., AND WELTY, C. A. Building watson: An overview of the deepqa project. *AI Magazine 31*, 3 (2010), 59–79.

[47] FONSECA, N. A., SANTOS COSTA, V., AND CAMACHO, R. Conceptual clustering of multi-relational data. In *Inductive Logic Programming: 21st International Conference, ILP 2011, Windsor Great Park, UK, July 31 – August 3, 2011, Revised Selected Papers* (Berlin, Heidelberg, 2012), S. H. Muggleton, A. Tamaddoni-Nezhad, and F. A. Lisi, Eds., Springer Berlin Heidelberg, pp. 145–159.

[48] FORTUNATO, S. Community detection in graphs. *Physics Reports 486*, 3-5 (2010), 75 – 174.

[49] FRASCONI, P., COSTA, F., DE RAEDT, L., AND DE GRAVE, K. klog: A language for logical and relational learning with kernels. *Artif. Intell. 217* (2014), 117–143.

[50] FRIESEN, A. L., AND DOMINGOS, P. Recursive decomposition for nonconvex optimization. In *Proceedings of the 24th International Conference on Artificial Intelligence* (2015), IJCAI'15, AAAI Press, pp. 253–259.

[51] GAIFMAN, H. On local and non-local properties. In *Proceedings of the Herbrand Symposium*, J. Stern, Ed., vol. 107 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1982, pp. 105 – 135.

[52] GARCÍA-DURÁN, A., DUMANČIĆ, S., AND NIEPERT, M. Learning sequence encoders for temporal knowledge graph completion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018* (2018), pp. 4816–4821.

[53] GAY, S., HARTERT, R., LECOUTRE, C., AND SCHAUS, P. Conflict ordering search for scheduling problems. In *Principles and Practice of Constraint*

Programming (Cham, 2015), G. Pesant, Ed., Springer International Publishing, pp. 140–148.

[54] GETOOR, L., AND TASKAR, B. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[55] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[56] GOODFELLOW, I. J., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Cambridge, MA, USA, 2014), NIPS'14, MIT Press, pp. 2672–2680.

[57] GREFENSTETTE, E. Towards a formal distributional semantics: Simulating logical calculi with tensors. *Proceedings of the Second Joint Conference on Lexical and Computational Semantics* (2013).

[58] HALPERN, J. Y. An analysis of first-order logics of probability. *Artificial Intelligence 46* (1990), 311–350.

[59] HASTAD, J. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1986), STOC '86, ACM, pp. 6–20.

[60] HASTAD, J., AND GOLDMANN, M. On the power of small-depth threshold circuits. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science* (Oct 1990), pp. 610–618 vol.2.

[61] HAUSSLER, D. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.

[62] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[63] HSU, F.-H. *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton University Press, Princeton, NJ, USA, 2002.

[64] HUANG, H.-C., CHUANG, Y.-Y., AND CHEN, C.-S. Affinity aggregation for spectral clustering. In *International Conference on Computer Vision and Pattern Recognition* (2012), IEEE Computer Society, pp. 773–780.

[65] JACKSON, P., AND SHERIDAN, D. Clause form conversions for boolean circuits. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing* (Berlin, Heidelberg, 2005), SAT'04, Springer-Verlag, pp. 183–198.

[66] JI, G., HE, S., XU, L., LIU, K., AND ZHAO, J. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (2015), Association for Computational Linguistics, pp. 687–696.

[67] KADLEC, R., BAJGAR, O., AND KLEINDIENST, J. Knowledge base completion: Baselines strike back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP, Rep4NLP at ACL 2017, Vancouver, Canada, August 3, 2017* (2017), pp. 69–74.

[68] KAHNEMAN, D. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, 2011.

[69] KARYPIS, G., AND KUMAR, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 1 (Dec. 1998), 359–392.

[70] KAUR, N., KUNAPULI, G., KHOT, T., KERSTING, K., COHEN, W., AND NATARAJAN, S. Relational restricted boltzmann machines: A probabilistic logic learning approach. In *Inductive Logic Programming* (Cham, 2018), N. Lachiche and C. Vrain, Eds., Springer International Publishing, pp. 94–111.

[71] KAZEMI, S. M., AND POOLE, D. Relnn: A deep neural model for relational learning. In *AAAI* (2018), AAAI Press.

[72] KHANNA, DIVYANSH ABD SAHU, R., BATHS, V., AND DESHPANDE, B. Comparative study of classification techniques (svm, logistic regression and neural networks) to predict the prevalence of heart disease. *International Journal of Machine Learning and Computing 5*, 5 (2015), 414–419.

[73] KINGMA, D. P., MOHAMED, S., JIMENEZ REZENDE, D., AND WELLING, M. Semi-supervised learning with deep generative models. In *NIPS*. Curran Associates, Inc., 2014, pp. 3581–3589.

[74] KINGMA, D. P., AND WELLING, M. Auto-encoding variational bayes. In *ICLR 2014* (apr 2014).

[75] KIRSTEN, M., AND WROBEL, S. Relational distance-based clustering. In *Lecture Notes in Computer Science* (1998), vol. 1446, Springer-Verlag, pp. 261–270.

[76] KOK, S., AND DOMINGOS, P. Statistical predicate invention. In *Proceedings of the 24th International Conference on Machine Learning* (New York, NY, USA, 2007), ICML '07, ACM, pp. 433–440.

[77] KOK, S., AND DOMINGOS, P. Extracting semantic networks from text via relational clustering. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I* (Berlin, Heidelberg, 2008), ECML PKDD '08, Springer-Verlag, pp. 624–639.

[78] KOK, S., AND DOMINGOS, P. Learning markov logic network structure via hypergraph lifting. In *ICML* (New York, NY, USA, 2009), ICML '09, ACM, pp. 505–512.

[79] KOK, S., AND DOMINGOS, P. Learning markov logic networks using structural motifs. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (2010), pp. 551–558.

[80] KOLLER, D., AND FRIEDMAN, N. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009.

[81] KRAMER, S. Predicate Invention: A Comprehensive View. *Technical report* (1995).

[82] LANDWEHR, N., PASSERINI, A., DE RAEDT, L., AND FRASCONI, P. kfoil: Learning simple relational kernels. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1* (2006), AAAI'06, AAAI Press, pp. 389–394.

[83] LAO, N., AND COHEN, W. W. Relational retrieval using a combination of path-constrained random walks. *Machine Learning 81*, 1 (Oct 2010), 53–67.

[84] LAO, N., MITCHELL, T., AND COHEN, W. W. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (Stroudsburg, PA, USA, 2011), EMNLP '11, Association for Computational Linguistics, pp. 529–539.

[85] LAVRAC, N., AND DZEROSKI, S. *Inductive Logic Programming: Techniques and Applications*. Routledge, New York, NY, 10001, 1993.

[86] LIN, Y., LIU, Z., SUN, M., LIU, Y., AND ZHU, X. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015), AAAI'15, AAAI Press, pp. 2181–2187.

[87] LOVÁSZ, L. Random walks on graphs: A survey. In *Combinatorics, Paul Erdős is Eighty*, D. Miklós, V. T. Sós, and T. Szőnyi, Eds., vol. 2. János Bolyai Mathematical Society, Budapest, 1996, pp. 353–398.

[88] MALL, R., LANGONE, R., AND SUYKENS, J. A. K. Multilevel Hierarchical Kernel Spectral Clustering for Real-Life Large Scale Complex Networks. mult. p. Comments: PLOS ONE, Vol 9, Issue 6, June 2014.

[89] MANHAEVE, R., DUMANČIĆ, S., KIMMIG, A., DEMEESTER, T., AND DE RAEDT, L. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31* (2018), S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., pp. 3753–3760.

[90] MIHALKOVA, L., AND MOONEY, R. J. Bottom-up learning of markov logic network structure. In *ICML 2007* (Corvallis, OR, June 2007).

[91] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119.

[92] MINERVINI, P., COSTABELLO, L., MUÑOZ, E., NOVÁCEK, V., AND VANDENBUSSCHE, P. Regularizing knowledge graph embeddings via equivalence and inversion axioms. In *ECML/PKDD (1)* (2017), vol. 10534 of *Lecture Notes in Computer Science*, Springer, pp. 668–683.

[93] MINERVINI, P., DEMEESTER, T., ROCKTÄSCHEL, T., AND RIEDEL, S. Adversarial sets for regularising neural link predictors. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017* (2017).

[94] MOREY, L. C., AND AGRESTI, A. The measurement of classification agreement: An adjustment to the rand statistic for chance agreement. *Educational and Psychological Measurement 44*, 1 (1984), 33–37.

[95] MUGGLETON, S., AND BUNTINE, W. L. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning (ICML'88)* (1988), J. Laird, Ed., Morgan Kaufmann, pp. 339–352.

[96] MUGGLETON, S., DAI, W., SAMMUT, C., TAMADDONI-NEZHAD, A., WEN, J., AND ZHOU, Z. Meta-interpretive learning from noisy images. *Machine Learning 107*, 7 (2018), 1097–1118.

[97] MUGGLETON, S., AND DE RAEDT, L. Inductive logic programming: Theory and methods. *J. Log. Program. 19/20* (1994), 629–679.

[98] MUGGLETON, S., AND LIN, D. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited.

[99] NEVILLE, J., ADLER, M., AND JENSEN, D. Clustering relational data using attribute and link information. In *Proceedings of the Text Mining and Link Analysis Workshop, 18th International Joint Conference on Artificial Intelligence* (2003), pp. 9–15.

[100] NG, A. Y., JORDAN, M. I., AND WEISS, Y. On spectral clustering: Analysis and an algorithm. In *Advances in neural information proicessing systems* (2001), MIT Press, pp. 849–856.

[101] NGUYEN, D. Q., SIRTS, K., QU, L., AND JOHNSON, M. Stranse: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (San Diego, California, June 2016), Association for Computational Linguistics, pp. 460–466.

[102] NICKEL, M., JIANG, X., AND TRESP, V. Reducing the rank in relational factorization models by including observable patterns. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1179–1187.

[103] NICKEL, M., TRESP, V., AND KRIEGEL, H.-P. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* (New York, NY, USA, 2011), L. Getoor and T. Scheffer, Eds., ACM, pp. 809–816.

[104] NIEPERT, M. Discriminative gaifman models. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (USA, 2016), NIPS'16, Curran Associates Inc., pp. 3413–3421.

[105] ONG, I. M., CASTRO DUTRA, I., PAGE, D., AND COSTA, V. S. Mode directed path finding. In *16th European Conference on Machine Learning* (Berlin, Heidelberg, 2005), Springer Berlin Heidelberg, pp. 673–681.

[106] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[107] PERLICH, C., AND PROVOST, F. Aggregation-based feature invention and relational concept classes. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2003), KDD '03, ACM, pp. 167–176.

[108] PERLICH, C., AND PROVOST, F. Distribution-based aggregation for relational learning with identifier attributes. *Mach. Learn. 62*, 1-2 (Feb. 2006), 65–105.

[109] PEROZZI, B., AL-RFOU, R., AND SKIENA, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2014), KDD '14, ACM, pp. 701–710.

[110] PFEIFFER, III, J. J., MORENO, S., LA FOND, T., NEVILLE, J., AND GALLAGHER, B. Attributed graph models: Modeling network structure with correlated attributes. In *Proceedings of the 23rd International Conference on World Wide Web* (New York, NY, USA, 2014), WWW '14, ACM, pp. 831–842.

[111] POOLE, D. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 2003), IJCAI'03, Morgan Kaufmann Publishers Inc., pp. 985–991.

[112] POPESCUL, A., AND UNGAR, L. H. Cluster-based concept invention for statistical relational learning. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2004), KDD '04, ACM, pp. 665–670.

[113] PUJARA, J., AUGUSTINE, E., AND GETOOR, L. Sparsity and noise: Where knowledge graph embeddings fall short. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2017).

[114] RAEDT, L. D., POOLE, D., KERSTING, K., AND NATARAJAN, S. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Morgan & Claypool Publishers, 2016.

[115] RAND, W. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66*, 336 (1971), 846–850.

[116] RANZATO, M., BOUREAU, Y.-L., AND LECUN, Y. Sparse feature learning for deep belief networks. In *NIPS* (2007), Curran Associates, Inc., pp. 1185–1192.

[117] RICHARDS, B. L., AND MOONEY, R. J. Learning relations by pathfinding. In *Proc. of AAAI-92* (San Jose, CA, 1992), pp. 50–55.

[118] RICHARDSON, M., AND DOMINGOS, P. Markov logic networks. *Mach. Learn. 62*, 1-2 (Feb. 2006), 107–136.

[119] ROBINSON, J. A. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (Jan. 1965), 23–41.

[120] ROCKTÄSCHEL, T., BOŠNJAK, M., SINGH, S., AND RIEDEL, S. Low-dimensional embeddings of logic. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing* (2014), Association for Computational Linguistics, pp. 45–49.

[121] ROCKTÄSCHEL, T., AND RIEDEL, S. End-to-end differentiable proving. In *NIPS*. Curran Associates, Inc., 2017, pp. 3788–3800.

[122] ROCKTÄSCHEL, T., SINGH, S., AND RIEDEL, S. Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015), Association for Computational Linguistics, pp. 1119–1129.

[123] ROSSI, F., BEEK, P. V., AND WALSH, T. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[124] ROUSSEEUW, P. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math. 20*, 1 (Nov. 1987), 53–65.

[125] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall Press, Upper Saddle River, NJ, USA, 2009.

[126] SCHLICHTKRULL, M. S., KIPF, T. N., BLOEM, P., VAN DEN BERG, R., TITOV, I., AND WELLING, M. Modeling relational data with graph convolutional networks. *CoRR abs/1703.06103* (2017).

[127] SEN, P., NAMATA, G. M., BILGIC, M., GETOOR, L., GALLAGHER, B., AND ELIASSI-RAD, T. Collective classification in network data. *AI Magazine 29*, 3 (2008), 93–106.

[128] SHERVASHIDZE, N., AND BORGWARDT, K. Fast subtree kernels on graphs. In *Proceedings of the Neural Information Processing Systems Conference NIPS 2009*. Neural Information Processing Systems Foundation, 2009, pp. 1660–1668.

[129] SHERVASHIDZE, N., SCHWEITZER, P., VAN LEEUWEN, E. J., MEHLHORN, K., AND BORGWARDT, K. M. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res. 12* (Nov. 2011), 2539–2561.

[130] SILVER, D., HUANG, A., MADDISON, C. J., GUEZ, A., SIFRE, L., VAN DEN DRIESSCHE, G., SCHRITTWIESER, J., ANTONOGLOU, I., PANNEERSHELVAM, V., LANCTOT, M., DIELEMAN, S., GREWE, D., NHAM, J., KALCHBRENNER, N., SUTSKEVER, I., LILLICRAP, T., LEACH, M., KAVUKCUOGLU, K., GRAEPEL, T., AND HASSABIS, D. Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 7587 (jan 2016), 484–489.

[131] SILVER, D., SCHRITTWIESER, J., SIMONYAN, K., ANTONOGLOU, I., HUANG, A., GUEZ, A., HUBERT, T., BAKER, L., LAI, M., BOLTON, A., CHEN, Y., LILLICRAP, T., HUI, F., SIFRE, L., VAN DEN DRIESSCHE, G., GRAEPEL, T., AND HASSABIS, D. Mastering the game of go without human knowledge. *Nature 550* (oct 2017), 354–.

[132] SILVERSTEIN, G., AND PAZZANI, M. J. Relational clich'es: Constraining constructive induction during relational learning. In *In Proceedings of the Eighth International Workshop on Machine Learning* (1991), Morgan Kaufmann, pp. 203–207.

[133] ŠOUREK, G., MANANDHAR, S., ŽELEZNÝ, F., SCHOCKAERT, S., AND KUŽELKA, O. Learning predictive categories using lifted relational neural networks. In *Inductive Logic Programming* (Cham, 2017), J. Cussens and A. Russo, Eds., Springer International Publishing, pp. 108–119.

[134] SRINIVASAN, A., MUGGLETON, S., AND BAIN, M. Distinguishing exceptions from noise in non-monotonic learning. In *In Proceedings of the 2nd International Workshop on Inductive Logic Programming* (1992), pp. 97–107.

[135] STEINHAUS, H. Sur la division des corps matériels en parties. *Bull. Acad. Pol. Sci., Cl. III 4* (1957), 801–804.

[136] SUGIYAMA, M., AND BORGWARDT, K. Halting in random walk kernels. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, pp. 1639–1647.

[137] SUYKENS, J. A., ALZATE, C., AND PELCKMANS, K. Primal and dual model representations in kernel-based learning. *Statist. Surv. 4* (2010), 148–183.

[138] SUYKENS, J. A. K. Deep restricted kernel machines using conjugate feature duality. *Neural Comput. 29*, 8 (Aug. 2017), 2123–2163.

[139] TOUTANOVA, K., AND CHEN, D. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality* (2015), pp. 57–66.

[140] TROUILLON, T., WELBL, J., RIEDEL, S., GAUSSIER, E., AND BOUCHARD, G. Complex embeddings for simple link prediction. In *International Conference on Machine Learning (ICML)* (2016), vol. 48, pp. 2071–2080.

[141] TURING, A. M. Computers & thought. MIT Press, Cambridge, MA, USA, 1995, ch. Computing Machinery and Intelligence, pp. 11–35.

[142] VAN CRAENENDONCK, T., AND BLOCKEEL, H. Using internal validity measures to compare clustering algorithms. In *AutoML Workshop at 32nd International Conference on Machine Learning, Lille, 11 July 2015* (July 2015), pp. 1–8.

[143] VAN CRAENENDONCK, T., DUMANČIĆ, S., AND BLOCKEEL, H. COBRA: A fast and simple method for active clustering with pairwise constraints. In *IJCAI* (2017), ijcai.org, pp. 2871–2877.

[144] VAN CRAENENDONCK, T., DUMANČIĆ, S., WOLPUTTE, E. V., AND BLOCKEEL, H. COBRAS: interactive clustering with pairwise queries. In *Symposium on Intelligence Data Analysis (IDA)* (2018), vol. 11191 of *Lecture Notes in Computer Science*, Springer, pp. 353–366.

[145] VAN CRAENENDONCK, T., MEERT, W., DUMANČIĆ, S., AND BLOCKEEL, H. COBRASTS: A new approach to semi-supervised clustering of time series. In *Discovery Science* (2018), vol. 11198 of *Lecture Notes in Computer Science*, Springer, pp. 179–193.

[146] VAN DEN BROECK, G. *Lifted Inference and Learning in Statistical Relational Models*. PhD thesis, KU Leuven, Jan. 2013.

[147] VAN EMDEN, M. H., AND KOWALSKI, R. A. The semantics of predicate logic as a programming language. *J. ACM 23*, 4 (Oct. 1976), 733–742.

[148] VAN HAAREN, J., VAN DEN BROECK, G., MEERT, W., AND DAVIS, J. Lifted generative learning of markov logic networks. *Machine Learning 103*, 1 (Apr 2016), 27–55.

[149] VENDRAMIN, L., CAMPELLO, R. J. G. B., AND HRUSCHKA, E. R. Relative clustering validity criteria: A comparative overview. *Statistical Analysis and Data Mining 3*, 4 (2010), 209–235.

[150] VIG, L., SRINIVASAN, A., BAIN, M., AND VERMA, A. An investigation into the role of domain-knowledge on the use of embeddings. In *Inductive Logic Programming* (Cham, 2018), N. Lachiche and C. Vrain, Eds., Springer International Publishing, pp. 169–183.

[151] VINCENT, P., LAROCHELLE, H., BENGIO, Y., AND MANZAGOL, P.-A. Extracting and composing robust features with denoising autoencoders. In *ICML* (New York, NY, USA, 2008), ICML '08, ACM, pp. 1096–1103.

[152] WACHMAN, G., AND KHARDON, R. Learning from interpretations: a rooted kernel for ordered hypergraphs. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007* (2007), pp. 943–950.

[153] WANG, Q., MAO, Z., WANG, B., AND GUO, L. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering 29*, 12 (Dec 2017), 2724–2743.

[154] WANG, W. Y., MAZAITIS, K., AND COHEN, W. W. A soft version of predicate invention based on structured sparsity. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (2015), pp. 3918–3924.

[155] WARD, J. H. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association 58*, 301 (1963), 236–244.

[156] WIRTH, R. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining* (2000), pp. 29–39.

[157] WITSENBURG, T., AND BLOCKEEL, H. Improving the accuracy of similarity measures by using link information. In *Foundations of Intelligent Systems - 19th International Symposium, ISMIS 2011, Warsaw, Poland, June 28-30, 2011. Proceedings* (2011), pp. 501–512.

[158] WOGULIS, J., AND LANGLEY, P. Improving efficiency by learning intermediate concepts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1* (San Francisco, CA, USA, 1989), IJCAI'89, Morgan Kaufmann Publishers Inc., pp. 657–662.

[159] YANG, B., TAU YIH, W., HE, X., GAO, J., AND DENG, L. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations* (2015).

[160] ZHAO, H., ROBLES-KELLY, A., AND ZHOU, J. On the use of the chi-squared distance for the structured learning of graph embeddings. In *Proceedings of the 2011 International Conference on Digital Image Computing: Techniques and Applications* (Washington, DC, USA, 2011), DICTA '11, IEEE Computer Society, pp. 422–428.

[161] ZUIDBERG DOS MARTIRES, P., AND DUMANČIĆ, S. Reactive probabilistic programming. In *the International Conference on Probabilistic Programming* (2018).

List of publications

Journal articles

DUMANČIĆ, S., AND BLOCKEEL, H. An expressive dissimilarity measure for relational clustering using neighbourhood trees. *Machine Learning 106*, 9-10 (Oct. 2017), 1523–1545

Peer-reviewed Conference and Workshop Articles

DUMANČIĆ, S., AND BLOCKEEL, H. Clustering-based relational unsupervised representation learning with an explicit distributed representation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17* (2017), pp. 1631–1637

DUMANČIĆ, S., AND BLOCKEEL, H. Demystifying relational latent representations. In *Inductive Logic Programming* (2018), N. Lachiche and C. Vrain, Eds., Springer International Publishing, pp. 63–77. Best Student Paper Award

DUMANČIĆ, S., GARCIA-DURAN, A., AND NIEPERT, M. On embeddings as alternative paradigm for relational learning. In *Proceedings of the 8th International Worshop on Statistical Relational Artificial intelligence* (2018)

VAN CRAENENDONCK, T., DUMANČIĆ, S., AND BLOCKEEL, H. COBRA: A fast and simple method for active clustering with pairwise constraints. In *IJCAI* (2017), ijcai.org, pp. 2871–2877

DUMANČIĆ, S., AND BLOCKEEL, H. Unsupervised relational representation learning via clustering: Preliminary results. In *6th workshop on Statistical Relational Artificial Intelligence StarAI at IJCAI* (2016)

VAN CRAENENDONCK, T., DUMANČIĆ, S., WOLPUTTE, E. V., AND BLOCKEEL, H. COBRAS: interactive clustering with pairwise queries. In *Symposium on Intelligence Data Analysis (IDA)* (2018), vol. 11191 of *Lecture Notes in Computer Science*, Springer, pp. 353–366

VAN CRAENENDONCK, T., MEERT, W., DUMANČIĆ, S., AND BLOCKEEL, H. COBRASTS: A new approach to semi-supervised clustering of time series. In *Discovery Science* (2018), vol. 11198 of *Lecture Notes in Computer Science*, Springer, pp. 179–193

MANHAEVE, R., DUMANČIĆ, S., KIMMIG, A., DEMEESTER, T., AND DE RAEDT, L. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31* (2018), S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., pp. 3753–3760

GARCÍA-DURÁN, A., DUMANČIĆ, S., AND NIEPERT, M. Learning sequence encoders for temporal knowledge graph completion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018* (2018), pp. 4816–4821

Peer-reviewed Abstracts and Posters

ZUIDBERG DOS MARTIRES, P., AND DUMANČIĆ, S. Reactive probabilistic programming. In *the International Conference on Probabilistic Programming* (2018)

DUMANČIĆ, S., GUNS, T., MEERT, W., AND BLOCKEEL, H. Auto-encoding logic programs. In *Proceedings of the 2nd Workshop on Neural Abstract Machines and Program Induction* (2018)

DUMANČIĆ, S., MEERT, W., AND BLOCKEEL, H. Theory reconstruction: a representation learning view on predicate invention. In

6th workshop on Statistical Relational Artificial Intelligence StarAI at IJCAI (2016)

DUMANČIĆ, S., AND BLOCKEEL, H. An efficient and expressive similarity measure for relational clustering using neighbourhood trees. In *ECAI*, vol. 285 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 1674–1675



FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DECLARATIVE LANGUAGES AND ARTIFICIAL INTELLIGENCE
Celestijnenlaan 200A box 2402
B-3001 Leuven
sebastijan.dumancic@cs.kuleuven.be
https://sebdumancic.github.io/