

SAŠO DŽEROSKI
NADA LAVRAČ
EDITORS

Relational Data Mining



Springer

GOOGLE EXHIBIT 1008
Google v. K.Mizra
IPR2026-00304

As the first book devoted to relational data mining, this coherently written multi-author monograph provides a thorough introduction and systematic overview of the area. The first part introduces the reader to the basics and principles of classical knowledge discovery in databases and inductive logic programming; subsequent chapters by leading experts assess the techniques in relational data mining in a principled and comprehensive way; finally, three chapters deal with advanced applications in various fields and refer the reader to resources for relational data mining.

This book will become a valuable source of reference for R&D professionals active in relational data mining. Students as well as IT professionals and ambitious practitioners interested in learning about relational data mining will appreciate the book as a useful text and gentle introduction to this exciting new field.

ISBN 3-540-42289-7



9 783540 422891

<http://www.springer.de>

Sašo Džeroski • Nada Lavrač (Eds.)

Relational Data Mining

With 79 Figures and 69 Tables



Springer

LINDA HALL LIBRARY
Kansas City, Mo.

QA 76.9
. D3R435
2001

Editors

Sašo Džeroski
Nada Lavrač

Jožef Stefan Institute
Jamova 39
1000 Ljubljana, Slovenia
E-mail: saso.dzeroski@ijs.si

Library of Congress Cataloging-in-Publication data applied for

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Relational data mining/Sašo Džeroski; Nada Lavrač (ed.). – Berlin;
Heidelberg; New York; Barcelona; Hong Kong; London; Milan;
Paris; Tokyo: Springer, 2001

ISBN 3-540-42289-7

ACM Subject Classification (1998): H.2.8, I.2.6, I.2.4, D.1.6, I.5

ISBN 3-540-42289-7 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
is a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

The use of general descriptive names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by the editors
Cover Design: d&p, design & production, Heidelberg
Printed on acid-free paper SPIN 10843997 – 06/3142SR – 5 4 3 2 1 0

Foreword

The area of data mining, or knowledge discovery in databases, started to receive a lot of attention in the 1990s. Developments in sensing, communications and storage technologies made it possible to collect and store large collections of scientific and industrial data. The abilities to analyze such data sets had not developed as fast. Data mining research arose at the intersection of several different research areas, notably statistics, databases, machine learning and algorithms. The area can loosely be defined as the analysis of large collections of data for finding models or patterns that are interesting or valuable.

The development of data mining methods requires the solution of several different types of problems. The data can have a very large number of dimensions, indicating that for example examining every pair of variables is impractical. The data can have hundreds of millions of observations, and therefore only a limited number of passes through the data can be done. The data can be observations of a process about which very little is known; hence there is no background knowledge available, and thus selection of appropriate models can be challenging. Or there can be heaps of background knowledge available, and methods that overlook it are destined to fail.

Most data mining methods have been developed for data in the traditional matrix form: rows represent observations, and columns represent variables. This representation has been the traditional one used in statistics, and it has many advantages. For example, matrix operations can be used to represent several data analytic procedures quite succinctly, and these representations make it possible to devise efficient algorithms.

However, data about the real world is seldom of this form. Rather, the application domain contains several different types of entities, of which different types of data are known. Only recently has a large body of research aimed at data mining on such data emerged.

Relational data mining studies methods for knowledge discovery in databases when the database has information about several types of objects. This, of course, is usually the case when the database has more than one table. Hence there is little doubt as to the relevance of the area; indeed, one can wonder why most of data mining research has concentrated on the single table case.

Table of Contents

Part I. Introduction

1. Data Mining in a Nutshell

Sašo Džeroski	3
1.1 Introduction	3
1.2 Data mining tasks	4
1.3 Patterns	8
1.4 Basic algorithms	16
1.5 Relational data mining	22
1.6 Data mining literature and Internet resources	25
1.7 Summary	26

2. Knowledge Discovery in Databases: An Overview

Usama Fayyad	28
2.1 Introduction	28
2.2 From transactions to warehouses to KDD	29
2.3 Why data mining?	31
2.4 KDD and data mining	32
2.5 Data mining methods: An overview	34
2.6 Applications in science data analysis	37
2.7 Research challenges for KDD	41
2.8 ILP and KDD: Prospects and challenges	43
2.9 Concluding remarks	45

3. An Introduction to Inductive Logic Programming

Sašo Džeroski and Nada Lavrač	48
3.1 Introduction	48
3.2 Logic programming and databases	49
3.3 Logic programming in a nutshell	51
3.4 The basic ILP task: Relational rule induction	55
3.5 Structuring the space of clauses	58
3.6 Searching the space of clauses	60
3.7 Bounding the search for clauses	62

3.8	Transforming ILP problems to propositional form	67
3.9	Relational data mining tasks addressed by ILP	69
3.10	ILP literature	70
3.11	Summary	71
 4. Inductive Logic Programming for Knowledge Discovery in Databases		
	Stefan Wrobel	74
4.1	Introduction	74
4.2	ILP: Relational analysis technology	81
4.3	ILP subgroup discovery: MIDOS	84
4.4	Using MIDOS and other ILP methods in KEPLER	93
4.5	Conclusion	99
<hr/>		
Part II. Techniques		
<hr/>		
5. Three Companions for Data Mining in First Order Logic		
	Luc De Raedt, Hendrik Blockeel, Luc Dehaspe, and Wim Van Laer	105
5.1	Introduction	105
5.2	Representation	106
5.3	ICL: Inductive classification logic	114
5.4	TILDE: Top-down induction of logical decision trees	116
5.5	CLAUDIEN: Clausal discovery	117
5.6	Practical use: Getting started	119
5.7	Sample application: Mutagenesis	133
5.8	An exercise	136
5.9	Conclusions and practical info	137
 6. Inducing Classification and Regression Trees in First Order Logic		
	Stefan Kramer and Gerhard Widmer	140
6.1	Introduction	140
6.2	Tree induction in logic	142
6.3	Structural classification and regression trees (S-CART): The top level algorithm	145
6.4	Growing a tree in first-order logic	145
6.5	Model selection by error/cost complexity pruning	150
6.6	First-order model trees	152
6.7	Applications	153
6.8	Related work	154
6.9	Conclusion	156

7. Relational Rule Induction with CPROGOL4.4:	
A Tutorial Introduction	
Stephen Muggleton and John Firth	160
7.1 Introduction	160
7.2 How to obtain CPROGOL4.4	161
7.3 Developing an input file for CPROGOL4.4	162
7.4 The theory	167
7.5 Estimating accuracy and significance	179
7.6 Declarative bias	182
7.7 Setting resource bounds	185
7.8 Debugging PROGOL input files	186
7.9 Summary	187
8. Discovery of Relational Association Rules	
Luc Dehaspe and Hannu Toivonen	189
8.1 Introduction	189
8.2 From association rules to query extensions	190
8.3 Evaluation measures	194
8.4 Declarative language bias	198
8.5 Query (extension) discovery with WARMR	201
8.6 A sample run	206
8.7 Discussion	208
9. Distance Based Approaches	
to Relational Learning and Clustering	
Mathias Kirsten, Stefan Wrobel, and Tamás Horváth	213
9.1 Introduction	213
9.2 A first-order distance measure	215
9.3 Instance-based learning with RIBL2	220
9.4 Hierarchical agglomerative clustering with RDBC	221
9.5 FORC - k -means for multirelational data	223
9.6 A case study in mRNA signal structures	225
9.7 Conclusion	230
<hr/>	
Part III. From Propositional to Relational Data Mining	
<hr/>	
10. How to Upgrade Propositional Learners to First Order	
Logic: A Case Study	
Wim Van Laer and Luc De Raedt	235
10.1 Introduction	235
10.2 Knowledge representation	236
10.3 The propositional learner CN2	240

10.4 Upgrading CN2	241
10.5 Some experimental results with ICL	252
10.6 Related work and conclusions	256
11. Propositionalization Approaches to Relational Data Mining	
Stefan Kramer, Nada Lavrač, and Peter Flach	262
11.1 Introduction	262
11.2 Background and definition of terms	265
11.3 An example illustrating a simple propositionalization	267
11.4 Feature construction for general-purpose propositionalization	271
11.5 Special-purpose feature construction	274
11.6 Related transformation approaches	277
11.7 A sample propositionalization method: Extending LINUS to handle non-determinate literals	279
11.8 Concluding remarks	286
12. Relational Learning and Boosting	
Ross Quinlan	292
12.1 Introduction	292
12.2 Boosting	293
12.3 FOIL	294
12.4 Overview of FFOIL	297
12.5 Boosting FFOIL	299
12.6 Experiments	300
12.7 Summary	304
13. Learning Probabilistic Relational Models	
Lise Getoor, Nir Friedman, Daphne Koller, and Avi Pfeffer	307
13.1 Introduction	307
13.2 Probabilistic models	309
13.3 Relational models	312
13.4 Probabilistic relational models	315
13.5 Learning PRMs	321
13.6 Experimental results	325
13.7 Discussion and related work	329
13.8 Extensions	331
13.9 Conclusions	333

Part IV. Applications and Web Resources

14. Relational Data Mining Applications:**An Overview**

Sašo Džeroski	339
14.1 Introduction	339
14.2 Drug design	340
14.3 Predicting mutagenicity and carcinogenicity	343
14.4 Predicting protein structure and function	346
14.5 Medical applications	349
14.6 Environmental applications	350
14.7 Mechanical engineering applications	353
14.8 Traffic engineering applications	356
14.9 Text mining, Web mining, and natural language processing ..	357
14.10 Business data analysis	358
14.11 Miscellaneous applications	359
14.12 Summary and discussion	360

15. Four Suggestions and a Rule**Concerning the Application of ILP**

Ashwin Srinivasan	365
15.1 Introduction	365
15.2 Background	366
15.3 When and why ILP?	367
15.4 Encoding background knowledge	368
15.5 Utility mismatch	371
15.6 Comprehensibility	371
15.7 From nursery slopes to Darwin's rule	372

16. Internet Resources on ILP for KDD

Ljupčo Todorovski, Irene Weber, Nada Lavrač, Olga Stěpánková,
Sašo Džeroski, Dimitar Kazakov, Darko Zupanič, and Peter Flach 375

16.1 Introduction	375
16.2 Brief history on ILP Internet resources	376
16.3 ILPnet2 Internet resources	377
16.4 Other ILP-related Internet resources	383
16.5 KDD related Internet resources	385
16.6 Conclusion	385

Author Index	389
---------------------------	-----

Subject Index	391
----------------------------	-----

9. Distance Based Approaches to Relational Learning and Clustering

Mathias Kirsten¹, Stefan Wrobel², and Tamás Horváth¹

¹ German National Research Center for Information Technology, GMD - AiS.KD
Schloß Birlinghoven, D-53754 Sankt Augustin, Germany

² School of Computer Science, IWS, University of Magdeburg
Universitätsplatz 2, D-39016 Magdeburg, Germany

Abstract

Within data analysis, *distance-based* methods have always been very popular. Such methods assume that it is possible to compute for each pair of objects in a domain their mutual distance (or similarity). In a distance-based setting, many of the tasks usually considered in data mining can be carried out in a surprisingly simple yet powerful way. In this chapter, we give a tutorial introduction to the use of distance-based methods for relational representations, concentrating in particular on predictive learning and clustering. We describe in detail one relational distance measure that has proven very successful in applications, and introduce three systems that actually carry out relational distance-based learning and clustering: RIBL2, RDBC and FORC. We also present a detailed case study of how these three systems were applied to a domain from molecular biology.

9.1 Introduction

Within data analysis, *distance-based* methods have always been very popular [9.23]. The central assumption of distance-based methods is that it is possible, for a particular domain under consideration, to specify for each pair of objects their mutual distance (or similarity). In certain settings, these distance values can be assumed to be given as part of the application, e.g., when humans are asked to rate distances between pairs of choices as done in certain psychological experiments. More often, however, in data analysis we are starting with objects described in a certain representation language (our *instance space* X) and use appropriate algorithms to compute distances for these objects.

Once we have such distances between object pairs, many of the tasks usually considered in data mining can be carried out in a disarmingly simple yet powerful way.

9.1.1 Predictive learning

Let us first consider how it is possible to perform predictive learning from examples given only distances between pairs of objects. Instead of forming an explicit hypothesis in the form of first-order clauses, we simply store all

available examples. Later, when we are asked to classify an unseen object, we compare this new object to all previously seen objects, and predict the function value associated with the *nearest neighbor* or our query object, i.e., the example object that has the smallest distance to our query object.

Considering only one object makes this algorithm very sensitive to noise, so generally, the method of choice is k -nearest neighbor, where the k nearest objects get to vote by majority on the prediction. Since this method of learning is strongly based on the instances available as examples, it is often referred to as *instance-based learning*. Many authors also call it *lazy learning*, emphasizing the fact that the learner does not bother to explicitly form a hypothesis, but lazily waits until an object to be classified comes along, and only then forms a (local, query-specific) hypothesis.

9.1.2 Clustering

The task of cluster analysis is, roughly speaking, the task of grouping a set of instances $\mathcal{I}$ from an instance space X into different subsets (clusters) such that objects belonging to the same cluster are maximally similar to each other while simultaneously being maximally dissimilar to objects from other clusters. Finding a clustering that is guaranteed to be optimal in terms of a chosen quality measure is an infeasible task, as it would require an exhaustive search through the space of all possible groupings of objects.

Hence, the available distance-based clustering algorithms use heuristic strategies, two of the more prominent of which are bottom-up *agglomerative* clustering strategies, and iterative *k-means* clustering. Hierarchical agglomerative clustering (c.f. [9.3]) constructs a hierarchy of nested clusterings by repeatedly merging the two existing clusters that are closest to each other, starting with all the individual instances as one-element clusters. The *k-means* method [9.15], on the other hand, first randomly selects k cluster centers, then repeatedly (a) assigns each instance to the cluster of the nearest cluster center, and (b) computes the new cluster center for each of the k current clusters, until a termination criterion is satisfied.

In this chapter, we give a tutorial introduction into the use of the above-described distance-based methods for learning and clustering. As a key component, in Section 9.2, we first explain what are the special challenges in defining a distance for first-order representations, and show in detail one first-order distance measure that has proven very successful in applications. We then turn to three systems that actually carry out first-order distance-based learning and clustering: RIBL2 (Section 9.3) is an instance-based learner, while RDBC and FORC (Sections 9.4 and 9.5) are agglomerative and k -means clustering systems, respectively. In each of the system's descriptions, we also provide more detail as to the workings of the chosen basic method, and discuss the particular adaptations to first-order problems. Finally, in Section 9.6, we present a detailed case study of how these three systems were applied to

Let us assume that $I = P(\text{set1}, 125, \text{personal})$ is an instance in our example and is further described by the ground atoms

```
cheese(set1, camembert, 150, france)
wine(set1, mouton, 1988, 0.75)
wine(set1, gallo, 1995, 0.5)
vineyard(gallo, famous, large, usa)
vineyard(mouton, famous, small, france) .
```

The set consisting of I and the above five ground atoms is the *case* of I .

Our example indicates that a *type* defining the set of possible values can be assigned to each argument of the predicate symbols. Furthermore, the case of an instance is usually not explicitly given. Instead, a set $\mathcal{B}$ of ground atoms, also referred to as *background knowledge*, is given, and for a given instance I , a subset of $\mathcal{B}$ is computed that defines (together with I) the case of I . This approach allows for a more succinct representation, as ground atoms may occur in more than one case.

More precisely, when applying one of our algorithms, the user has first to define the vocabulary of the predicate symbols to be used in the relational representation. Among these symbols there is a distinguished predicate symbol P that will be used to represent the elements of the set X of all possible *instances*, also called *instance space*. The other predicate symbols, called *background predicates*, are used to describe the user's background knowledge about the task. Each predicate symbol is associated (also by the user) with a declaration defining the types and modes of its arguments. The type of an argument is one of the basic types from $\{\text{number}, \text{discrete}, \text{list}, \text{term}, \text{name}\}$, while its mode defines whether or not the argument must be compared with corresponding arguments during the distance computation. The type 'name' is used for arguments that contain identifiers representing objects.

Example 9.2.2. The vocabulary of Example 9.2.1 consists of P and the background predicates 'cheese', 'wine', and 'vineyard' that are associated with the following declaration:

```
P(a1 : name, a2 : number, a3 : discrete)
cheese(a1 : name, a2 : discrete, a3 : number, a4 : discrete)
wine(a1 : name, a2 : discrete, a3 : number, a4 : number)
vineyard(a1 : name, a2 : discrete, a3 : discrete, a4 : discrete) .
```

In order to compute a case of an instance $I \in X$, we rely on the instance I , the background knowledge $\mathcal{B}$, and a non-negative integer $d \geq 0$ called *depth bound*. A case of I with respect to $\mathcal{B}$ and d is defined as the largest set $\{I\} \cup \mathcal{B}'$ such that $\mathcal{B}' \subseteq \mathcal{B}$ and for every background atom $B \in \mathcal{B}'$ it holds that there are $d' < d$ and $B_0, B_1, \dots, B_{d'-1}, B_{d'}$ satisfying

– $B_0 = I$, $B_{d'} = B$, and $B_i \in \mathcal{B}'$ for $i = 1, \dots, d' - 1$,

- B_i and B_{i+1} contain at least one common identifier for every $i = 0, \dots, d' - 1$.

Example 9.2.3. To illustrate the above definition let us assume that the background knowledge $\mathcal{B}$ in Example 9.2.1 consists of the following facts:

```
% set ID:name, cheese type:discrete, weight:number, origin:discrete
cheese(set1, camembert, 150, france)
cheese(set25, roquefort, 200, france)
cheese(set25, ricotta, 100, italy)
```

```
% set ID:name, vineyard ID:name, year:number, bottle size:number
wine(set1, mouton, 1988, 0.75)
wine(set1, gallo, 1995, 0.5)
wine(set25, mouton, 1995, 0.75)
```

```
% vineyard ID:name, popularity:discrete, size:number, country:discrete
vineyard(gallo, famous, large, usa)
vineyard(mouton, famous, small, france)
```

Then the case defined by the instance $P(\text{set1}, 125, \text{personal})$ with respect to $\mathcal{B}$ and $d = 2$ is given in Figure 9.1.

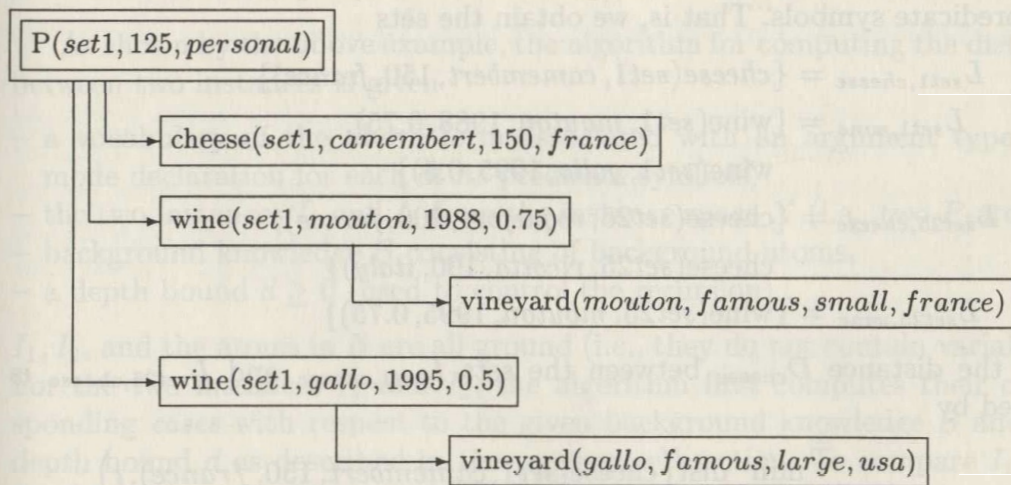


Fig. 9.1. The case defined by I_1 with respect to $\mathcal{B}$ and $d = 2$ in Example 9.2.1.

9.2.2 The distance measure

Before giving an informal description of the algorithm for computing the distance between two instances, we first give an example to illustrate how this algorithm works.

Example 9.2.4. Suppose we are given

- the instances $I_1 = P(\text{set1}, 125, \text{personal})$, $I_2 = P(\text{set25}, 195, \text{mail})$,
- the background knowledge $\mathcal{B}$ of Example 9.2.3,
- the depth bound $d = 2$,

and want to compute the distance between cases of I_1 and I_2 with respect to $\mathcal{B}$ and d .

This distance is computed as the normalized sum of the distances between the instances' corresponding arguments, i.e.,

$$\text{dist}(I_1, I_2, \mathcal{B}, d) = \frac{1}{3} \left(\text{dist}(\text{set1}, \text{set25}) + \text{dist}(125, 195) + \text{dist}(\text{personal}, \text{mail}) \right). \quad (9.1)$$

For computing the last two terms, standard propositional distance measures can be applied, i.e.,

$$\text{dist}(125, 195) = |195 - 125|/500 = 0.14 \quad (9.2)$$

(here the denominator denotes the highest possible price) and

$$\text{dist}(\text{personal}, \text{mail}) = 1 \quad (9.3)$$

(as they are different). In order to compute the distance between the object identifiers *set1* and *set25*, from the cases of *set1* and *set25* we first collect the sets of literals containing *set1* and *set25*, respectively, and partition them by the predicate symbols. That is, we obtain the sets

$$L_{\text{set1}, \text{cheese}} = \{\text{cheese}(\text{set1}, \text{camembert}, 150, \text{france})\}$$

$$L_{\text{set1}, \text{wine}} = \{\text{wine}(\text{set1}, \text{mouton}, 1988, 0.75), \\ \text{wine}(\text{set1}, \text{gallo}, 1995, 0.5)\}$$

$$L_{\text{set25}, \text{cheese}} = \{\text{cheese}(\text{set25}, \text{roquefort}, 200, \text{france}), \\ \text{cheese}(\text{set25}, \text{ricotta}, 100, \text{italy})\}$$

$$L_{\text{set25}, \text{wine}} = \{\text{wine}(\text{set25}, \text{mouton}, 1995, 0.75)\}$$

Now the distance D_{cheese} between the sets $L_{\text{set1}, \text{cheese}}$ and $L_{\text{set25}, \text{cheese}}$ is defined by

$$D_{\text{cheese}} = \frac{\min_{l \in L_{\text{set25}, \text{cheese}}} \text{dist}(\text{cheese}(\text{set1}, \text{camembert}, 150, \text{france}), l)}{|L_{\text{set25}, \text{cheese}}|} \quad (9.4)$$

$$= \frac{\min \left(\left(1 + \frac{|150 - 200|}{300} + 0\right)/3, \left(1 + \frac{|150 - 100|}{300} + 1\right)/3 \right)}{2} \quad (9.5)$$

$$= 0.1944 .$$

We note that in (9.4) we normalize by the cardinality of the *larger* set $L_{\text{set25}, \text{cheese}}$, and by 3 the two terms in (9.5), as we do not compare again the first arguments containing the set identifiers. Applying that

$$\begin{aligned}
\text{dist}(\text{mouton}, \text{gallo}) &= \\
&= \frac{\min \left(\frac{\text{dist}(\text{famous}, \text{famous}) + \text{dist}(\text{small}, \text{large}) + \text{dist}(\text{france}, \text{usa})}{3} \right)}{1} \\
&= 0.6666 ,
\end{aligned}$$

the distance D_{wine} between $L_{\text{set1}, \text{wine}}$ and $L_{\text{set25}, \text{wine}}$ is computed in a similar manner, i.e.,

$$\begin{aligned}
D_{\text{wine}} &= \frac{\min_{l \in L_{\text{set1}, \text{wine}}} \text{dist}(\text{wine}(\text{set25}, \text{mouton}, 1995, 0.75), l)}{|L_{\text{set1}, \text{wine}}|} \\
&= \frac{\min \left((0 + \frac{|1995-1988|}{100} + 0)/3, (0.6666 + 0 + \frac{|0.75-0.5|}{1})/3 \right)}{2} \\
&= 0.0117 .
\end{aligned}$$

Using the above values, the distance between object identifiers *set1* and *set25* is then computed by

$$\begin{aligned}
\text{dist}(\text{set1}, \text{set25}) &= \frac{D_{\text{cheese}} + D_{\text{wine}}}{2} \\
&= 0.1031 .
\end{aligned}$$

Combining (9.2), (9.3), and the above equation with (9.1), we finally obtain

$$\text{dist}(I_1, I_2, \mathcal{B}, d) = (0.1031 + 0.14 + 0)/3 = 0.081 .$$

As shown by the above example, the algorithm for computing the distance between two instances is given

- a vocabulary of the above form associated with an argument type and mode declaration for each of its predicate symbols,
- the two instances I_1 and I_2 from the instance space X (i.e., two P -atoms),
- background knowledge $\mathcal{B}$ consisting of background atoms,
- a depth bound $d \geq 0$ (used to control the recursion) .

I_1, I_2 , and the atoms in $\mathcal{B}$ are all ground (i.e., they do not contain variables). For the two instances I_1 and I_2 , the algorithm first computes their corresponding *cases* with respect to the given background knowledge $\mathcal{B}$ and the depth bound d as described in the previous subsection. To compare I_1 and I_2 , the distance function recursively descends into their corresponding cases, beginning with the ground P -atoms I_1 and I_2 . In general, when comparing two ground atoms, each with the same predicate symbol, one first computes the distance between their corresponding arguments that have been marked for comparison in the mode declaration, then sums up these distances, and normalizes finally this sum with the number of marked arguments. Depending on their common type, two arguments are compared as follows.

number and discrete: For arguments of these basic types, some corresponding elementary distance function is called.

list and term: For arguments of these types, our measure employs edit distances. Intuitively, with an edit distance, we are given a set of edit operations such as insert, delete, and change with an associated cost function that tells us how expensive it is to delete or insert an element of the underlying alphabet, or how expensive it is to change one into another. The edit distance is then defined as the smallest cost of a sequence of such operations that turns the first object into the second. For both of lists and terms, there are efficient algorithms for computing the edit distance (see, e.g., [9.22, 9.24]).

name: When comparing two arguments of this type, the algorithm first checks whether the depth of the recursion has already reached the input parameter d (i.e., the depth bound d is used to ensure limited effort for the recursive computation). If yes, the arguments are compared as that of type discrete. Otherwise, all further facts from the cases about the respective named objects are first collected. Each of these two sets is then partitioned by the predicate symbols, and for every predicate symbol we compare its corresponding subsets from the two sets as follows. For each fact from the smallest set we compute recursively its distance from the larger set (i.e., the distance between this fact and its nearest fact from the larger set), then sum up these distances and normalize with the cardinality of the larger set.

9.3 Instance-based learning with RIBL2

RIBL2 is a first-order instance-based learner applying the classical k -nearest neighbor (kNN) classifier in a relational representation, using the distance measure described in the preceding section. In more detail, a k -nearest neighbor algorithm determines its predictions as follows. To simplify the discussion, note that our distance measure takes on values in the range $[0, 1]$, so we can equivalently work with the function

$$\text{sim}(x, y) := 1 - \text{dist}(x, y)$$

which is 1 for identical objects and falls towards 0 with increasing distance between objects. Given a set of examples E and a new instance x_{new} from X to be classified, let us call $N \subseteq E$ the set of k nearest neighbors of x_{new} according to sim . Further details depend on whether we are predicting a discrete or continuous variable. For a *discrete* variable, denote by

$$N_{y_j} := \{e = (x, y_j) \in N\}.$$

those neighbors that are voting for value y_j . The algorithm will then predict

$$h(x_{\text{new}}) := \text{argmax}_{y_j} \sum_{e=(x, y_j) \in N_{y_j}} \text{sim}(x, x_{\text{new}})^a,$$

where a is a parameter that determines how much an influence is allocated to farther-away neighbors. If we are predicting a continuous variable, we use a weighted average instead:

$$h(x_{new}) := \frac{\sum_{e=(x,y) \in N} \text{sim}(x, x_{new})^a \cdot y}{\sum_{e=(x,y) \in N} \text{sim}(x, x_{new})^a}.$$

In order to determine k , most k -nearest neighbor algorithm internally estimate the accuracy that can be achieved (e.g., by cross-validation), and then choose the value of k that has ranked highest for all future classifications. Despite this very simple principle, k -nearest neighbor methods are known to be competitive with and often superior to other approaches in terms of predictive accuracy. Thus, they are attractive choices whenever an explicit hypothesis is not needed.

9.4 Hierarchical agglomerative clustering with RDBC

Finding a clustering that is guaranteed to be optimal in terms of a chosen quality measure is an infeasible task, as it would require an exhaustive search through the space of all possible clusterings. Hence, the available distance-based clustering algorithms use heuristic strategies. The RDBC algorithm employs one of the most basic distance-based clustering methods, namely hierarchical agglomerative clustering (c.f. [9.3]).

9.4.1 Forming a cluster hierarchy

Together with the distance measure described in Section 9.2, hierarchical agglomerative clustering does not build flat partitions of the data but iteratively constructs a list of nested clusterings $C_1, \dots, C_n$. Figure 9.2 shows an example together with the typical visualization of such a clustering in a *dendrogram*.

The construction of the hierarchical clustering is done in a bottom-up manner by starting with the clustering C_1 where each case forms its own one-elemental cluster. The next clustering is built by combining the two closest respectively most similar clusters to a new cluster. In this case, the distance between two clusters is simply the distance between the two cases, as each cluster consists of a single element. For all subsequent clusterings the step of combining the two most similar clusters is repeated until finally the clustering (C_n) consists of a single cluster only. When joining two clusters to a new one, the distances between the new cluster and all other clusters can be computed from the already known distances of the current clusters. Depending on the chosen computation rule we get different variants of the hierarchical agglomerative clustering:

$$\text{dist}(C_1 \cup C_2, C_3) := f(\text{dist}(C_1, C_3), \text{dist}(C_2, C_3)).$$

RDBC includes the four most popular variants: $f = \min$, the so called *single-link* clustering, $f = \max$ the *complete-link* clustering, and for $f = \text{average}$ the *average-link* clustering. A somewhat more complicated choice of f results in *Ward's method*, which guarantees that the sum of (squared) distances is minimized for each aggregation step (this does not necessarily hold for the final result though).

So, suppose we have a set of instances $I_1, \dots, I_5$ together with their pairwise distance information computed by RIBL's distance measure:

	I_1	I_2	I_3	I_4	I_5
I_1	0	0.12	0.60	0.64	0.77
I_2		0	0.70	0.80	0.84
I_3			0	0.35	0.45
I_4				0	0.40
I_5					0

Employing the *complete-link* variant of RDBC on this data yields the cluster hierarchy shown in Figures 9.2 and 9.3.

9.4.2 Turning the cluster hierarchy into flat partitions

The chosen agglomerative approach renders a distance-based hierarchy of cases where the distance between subtrees decreases as we move from root towards the leaves. For many applications, such a hierarchical tree is all that is required. If however, the clustering task requires the selection of a single set of clusters, we need a technique for turning the tree into such a set. Since we are trying to minimize intra-cluster distance (respectively maximize intra-cluster similarity), it is natural to base cluster selection from the tree on this criterion. RDBC therefore uses a search procedure that "cuts up" the tree according to growing intra-cluster similarity thresholds, and selects the threshold and its corresponding group of clusters that minimizes average intra-cluster distance. Thus the exact number of clusters does not have to be predefined by the user but is automatically computed according to the quality function. More precisely, in RDBC the quality of a set of clusters $\mathcal{C}$ is defined as:

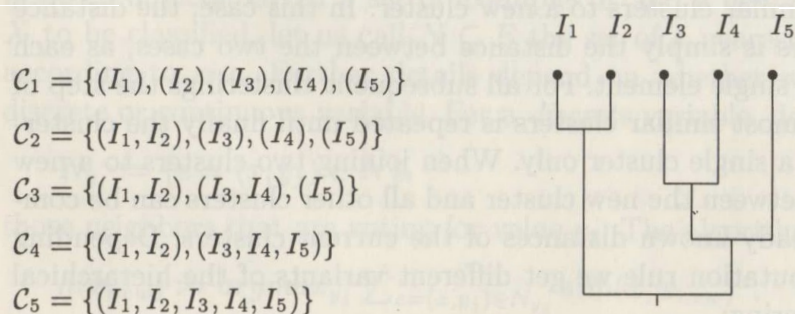


Fig. 9.2. Hierarchical clustering and its visualization in a dendrogram.

$$qual(C) = \frac{\sum_{C \in \mathcal{C}} dist_{intra}(C)}{|C|},$$

where $dist_{intra}(C)$ denotes the intra-cluster distance of a cluster C , which is the averaged sum of all distances between element pairs of the cluster, and is hence a measure of the cluster's compactness.

It should be noted, that the above quality measure cannot be applied directly as it reaches its maximum for a partitioning where each cluster consists of a single case. RDBC solves the problem by moving all single-element clusters into an *outlier cluster*, and the optimization of $qual(C)$ takes place on the resulting clustering.

So, coming back to our example involving $I_1, \dots, I_5$, the transformation of the hierarchy into a set of partitions firstly requires to find a threshold for which the above quality function $qual(C)$ is maximal.

All potential thresholds are evaluated by removing those connections from the hierarchical tree structure which lay below the given threshold. Thus, the result is a clustering forest rather than a tree. A cluster (respectively a partition) is then the set of instances belonging to a single tree¹.

In the example, cutting up the hierarchy at $d = 0.5$ results in two partitions $C_1 = \{I_1, I_2\}$ and $C_2 = \{I_3, I_4, I_5\}$. The quality for this clustering is calculated as

$$qual(\{C_1, C_2\}) = \frac{dist(I_1, I_2) + (dist(I_3, I_5) + dist(I_4, I_5) + dist(I_4, I_3)) / 3}{2} = 0.26$$

Finding the best threshold in the example is left as an exercise to the reader.

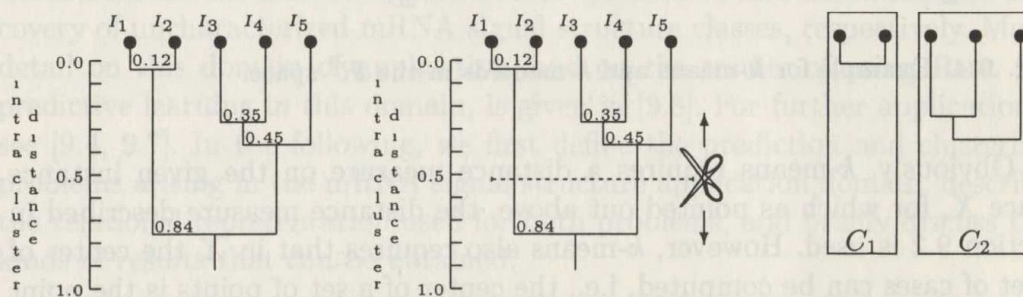


Fig. 9.3. Transforming a hierarchical clustering into flat partitions.

9.5 FORC - k -means for multirelational data

A very popular clustering technique for finding clusterings of a predefined size k is the k -means method [9.15]. FORC is the first clustering system to

¹ Isolated instances without any connections are not handled as one-elemental clusters but are gathered into the *outlier cluster*.

carry this algorithm over to first-order, relying on the distance measure of Section 9.2.

The basic idea of k -means is as follows. Firstly, an initial state has to be constructed by selecting or creating k cluster centers from X . In practice this is mostly done by randomly drawing k elements from the set of cases. With this initial state a two-step iteration is started:

1. Assign each case to the cluster of the nearest cluster center.
2. Compute the new cluster center for each of the k current cluster.
3. Go to step 1 unless clustering quality does not increase any more.

Thus, in each iteration the centers of the current clusters are determined in order to subsequently rebuild the k clusters on basis of the new centers. Figure 9.4 shows an example of this process for $X = \mathbb{R}^2$ (the corresponding cluster centers are pictured by the large circle respectively box).

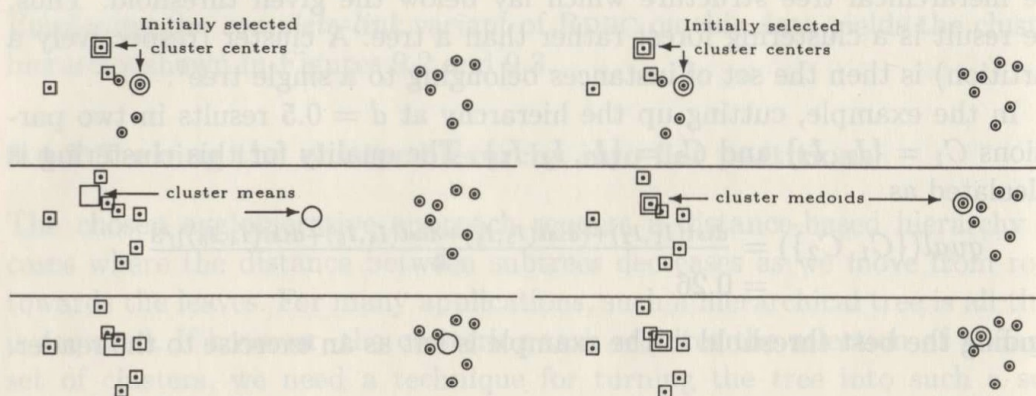


Fig. 9.4. Example for k -means and k -medoids in the $\mathbb{R}^2$ space.

Obviously, k -means requires a distance measure on the given instance space X , for which as pointed out above, the distance measure described in Section 9.2 is used. However, k -means also requires that in X the center of a set of cases can be computed, i.e., the center of a set of points is the point whose summed distances to the cases of the set is minimal. This is most easy if $X = \mathbb{R}^n$, i.e., for strictly numerical instance spaces. There the center of a set $C \subseteq X$ is simply given by its mean:

$$\text{center}(C) := \frac{1}{|C|} \sum_{c \in C} c$$

The corresponding (and usual) distance measure is the n -dimensional Euclidean metric but the Mahalanobis distance is [9.6, 9.14] also used.

For first-order data the problem is more challenging, since similarly obvious solutions for finding a center are not at hand. In fact, on multirelational data there is no longer any geometric or intuitive notion of the phrase “cluster

center". In addition, an analytic solution for the construction of the cluster center does not exist, therefore we have to make do with approximation.

FORC employs a solution that alleviates the problem of how to construct a previously not existing cluster center, which is to take an existing case as an approximation for the cluster center. This should be the most central instance of the set, i.e., the instance whose sum of squared distances to all other instances in the set is minimal and which is thus closest to the (unknown) cluster center. This approach is already known as the *k*-medoids method [9.10, 9.11], and first results of FORC obtained on the mutagenesis data set [9.21] using RIBL's distance measure show *k*-medoids' promising potential on first-order data [9.13].

Obviously, the whole process relies on distance information only, and is hence applicable to any representation for which an appropriate distance measure is available (e.g., a first-order representation + RIBL's distance measure). On the downside, when making no prior assumptions about the representation used, the computational complexity suffers from the expensive search for the medoid which requires time proportional to the squared number of instances in a cluster.

9.6 A case study in mRNA signal structures

In this section we present a case study designed to illustrate how relational distance-based learning and clustering algorithms can be applied in a real-world domain: the mRNA signal structure prediction and the automatic discovery of uncharacterized mRNA signal structure classes, respectively. More detail on this domain of application, and on the results of using RIBL for predictive learning in this domain, is given in [9.8]. For further applications, see [9.4, 9.7]. In the following, we first define the prediction and clustering problems arising in the mRNA signal structure application domain, describe the relational representation used for both problems, and briefly discuss the kinds of results that can be obtained.

9.6.1 The mRNA prediction and clustering problems

During the process of protein synthesis, messenger RNA (mRNA) molecules, a special kind of RNA molecules, mediate between the sites of information storage (i.e., DNA, which is functionally inactive) and protein synthesis. In general, an RNA is a linear sequence of the four bases (adenine – A, cytosine – C, guanine – G, and uracil – U) folded and coiled into specific conformations. The three-dimensional conformation of an RNA contains local regions of short complementary base-pairing, i.e., the two stable Watson-Crick base pairs A–U, G–C, and the weak G–U. The structure of RNA molecules is usually described at the following levels of abstraction:

primary structure: It is the linear sequence of the four bases that make up an RNA.

secondary structure: It is the collection of the base pairs occurring in the RNA's three-dimensional structure.

tertiary structure: It is the three-dimensional conformation of the RNA.

In addition, for the primary structures of messenger RNA molecules it also holds that they can be divided into three parts; they consist of a coding region (in their middle) defining the amino acid sequence of the target protein to be synthesized, and two non-coding regions (in their two ends) that contain regulatory motifs. Such regulatory motifs, called *signal structures*, control gene expression at the posttranscriptional level of the protein synthesis. Depending on their regulatory roles during the process of protein synthesis, signal structures are grouped into different signal structure *classes*. In this field of research, there are (at least) the following two challenging computational tasks:

- Detect (possibly new) signal structures belonging to known signal structure classes on a given uncharacterized mRNA.
- Find uncharacterized signal structure classes.

As a first step towards the direction of finding good algorithmical solutions to the above two problems, we consider the following two related problems, respectively. (In the description below, X denotes the set of all signal structures.)

signal structure prediction task: Given a set C of signal structure classes, a set $i \subseteq X$ of characterized signal structures (i.e., each with known class) belonging to one of the classes from C , and a signal structure $I \in X$ of unknown class from C , the aim is to give a good prediction of the class of I .

automatic discovery of uncharacterized signal structure classes:

In this task, we are given a set $i \subseteq X$ of uncharacterized signal structures (i.e., with unknown class information), the aim is to find a clustering $(I_1, \dots, I_k)$ of i with k as small as possible such that each cluster is a subset of some (possibly uncharacterized) signal structure class.

Now we turn to the problem of how to represent the components of the above computational tasks. When viewing the secondary structures of mRNA molecules, one can observe a set of different patterns [9.25], like hairpin loops, internal loops, etc, that make them up (see also Figure 9.5). These patterns are also referred to as *structure elements*. Furthermore, it also holds that mRNA's secondary structures can be viewed as trees (for more details see e.g., [9.20]), where the nodes represent the structure elements. As signal structures are (continuous) subsequences of the primary structures, this observation holds for them as well.

As an example, we consider a signal structure belonging to the class Selenocysteine Insertion Sequence (SECIS). Its primary structure is

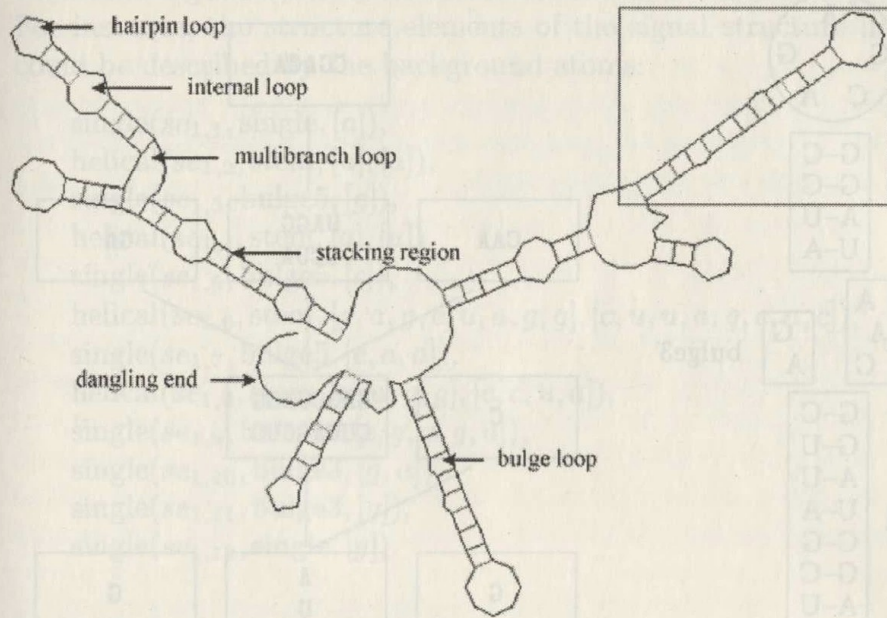


Fig. 9.5. mRNA structure with marked structure elements.

AUGACGAGCUAGGCAAUAGGCGAGACCUAGACUUAGCUCUGAG ,

and its secondary structure is given by the following complementary base pairs

```

AUGACGAGCUAGGCAAUAGGCG
| | | | | | | | | | | | | |
GAGU CUCGAUUCAG AUCCAG

```

In Figure 9.6, we give the structure elements that make up this secondary structure, and also the tree structure of these structure elements. As shown in the figure, the inner nodes on the longest path from the root all correspond to the pattern 'stem'. The root itself on this path is a dummy node, i.e., it does not represent any part of the signal structure. The other endpoint of the path corresponds to the hairpin loop (i.e., CGAGA) in the signal structure. This example indicates the following description of signal structures in our relational representation language.

The instances of this application are the signal structures. Using their tree structures, a signal structure, an instance from X , is described by the binary predicate

`structure(struct_id, tree_structure)`

where the first argument of type 'name' defines its identifier, and the second argument of type 'term' is used to describe its tree structure. For instance, the signal structure given in Figure 9.6 can be described by the ground atom

`structure(s1, dummy(se1,1, se1,2(se1,3, se1,4(...), se1,11), se1,12))`

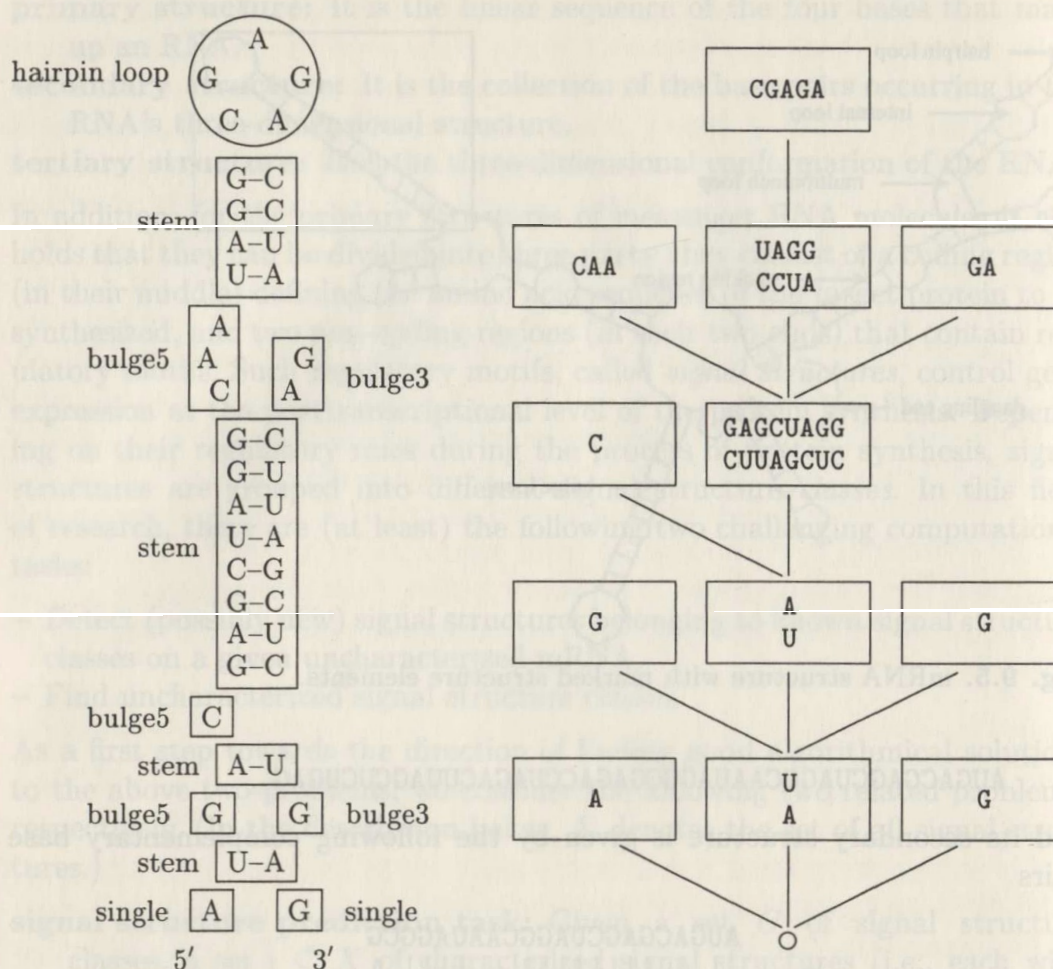


Fig. 9.6. The representation and the tree structure of a SECIS signal structure.

where 'dummy' is a constant denoting the root for each signal structure, and $se_{1,1}, \dots, se_{1,12}$ are further identifiers denoting the structure elements of s_1 . (We note that the identifiers $se_{1,1}, \dots, se_{1,12}$ are assigned to the descendants of the root of the tree given in Figure 9.6 according to preorder tree traversal.) The structure elements of the given signal structures are further described in the background knowledge. Depending on their types, structure elements are either helical (e.g., stems) or single (e.g., hairpin loop, bulge5, bulge3). A pattern of type single is described by the background predicate

$single(se_id, type, sequence)$

where the argument se_id of type 'name' is an identifier denoting the structure element, $type$ of type 'discrete' gives the structure element's pattern, and $sequence$ of type 'list' defines the primary structure of the structure element. Patterns of type helical are described by the background predicate

$helical(se_id, type, sequence_1, sequence_2)$

where the arguments have similar semantics as in the case of predicate single. For instance, the structure elements of the signal structure in our example could be described by the background atoms:

```

single(se1,1, single, [a]),
helical(se1,2, stem, [u], [a]),
single(se1,3, bulge5, [g]),
helical(se1,4, stem, [a], [u]),
single(se1,5, bulge5, [c]),
helical(se1,6, stem, [g, a, g, c, u, a, g, g], [c, u, u, a, g, c, u, c]),
single(se1,7, bulge5, [c, a, a]),
helical(se1,8, stem, [u, a, g, g], [c, c, u, a]),
single(se1,9, hairpin, [c, g, a, g, a]),
single(se1,10, bulge3, [g, a]),
single(se1,11, bulge3, [g]),
single(se1,12, single, [g]).

```

9.6.2 Results

For our experiments we have chosen a small domain consisting of 66 signal structures over 5 signal structure classes. For predictive learning, the optimal number of neighbors (k) has been estimated by *leave-one-out* cross-validation. The results for the different values of k are given in the following table (for more details see also [9.8]).

Table 9.1. Results for estimating the optimal number of neighbors (k).

#instances	#correctly classified for										best accuracy		
	$k =$	1	2	3	4	5	6	7	8	9	10	k	(%)
66		63	63	59	60	62	60	60	61	60	61	1	95.4

Clustering experiments were conducted with RDBC. Using all 66 instances for clustering, RDBC returned ten clusters (see Figure 9.7). While some of the actual classes were rediscovered by RDBC (*s_tar*, *s_ire*, *s_histone*, and *s_secise2*), the clustering of class *s_secise1* appears to be hard. Whereas the mixed cluster C_8 made up of *s_secise1*, and *s_secise2* may be caused by a folding which those instances have in common, cluster C_6 is hard to explain. In addition the misplaced instances *s_secise1*(5) and *s_secise1*(10) in cluster C_2 would better fit into cluster C_5 .

To further evaluate the quality of RDBC's results, we used the computed clusters for prediction. Two experiments were conducted and the prediction accuracy was determined via a *leave-one-out*-approach. In both experiments class information was not used during the clustering stage. In the first experiment all instances except the test instance were available for building the

clusters. The test instance's class was then determined by a majority vote over instances of the most similar of the obtained clusters (according to the distance measure). In contrast to that, for the second setup we included the test instance during clustering. As in the first experiment, we employed majority voting for determining the test instance's class². Table 9.2 gives the accuracies obtained by our two experiments.

Table 9.2. RDBC's accuracies on the mRNA domain.

RDBC with test instance	RDBC without test instance
87.9%	83.3%

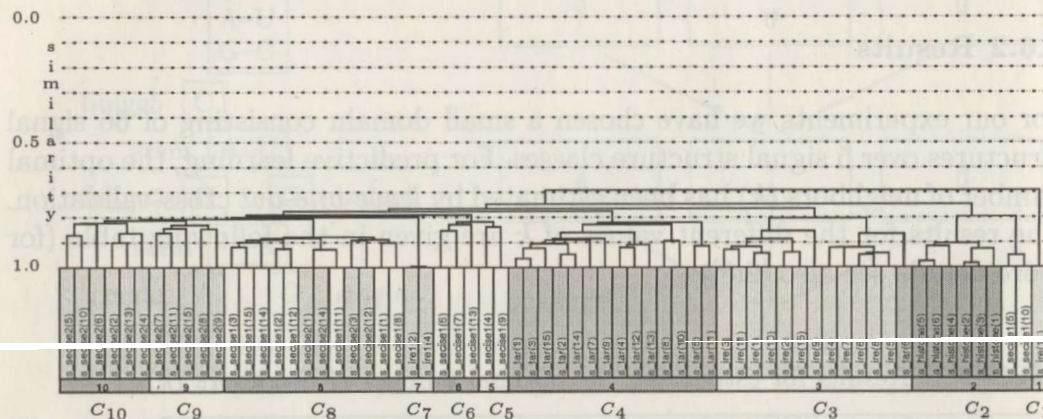


Fig. 9.7. Sample cluster hierarchy on the mRNA data. Shaded bars at the bottom mark the clusters found by RDBC.

9.7 Conclusion

First-order distance-based learning and clustering methods offer a lot of potential for applications involving complex and structured objects. For learning, distance-based “lazy” methods are known to often outperform their non-lazy counterparts; for clustering, a distance-based formulation is naturally at the heart of cluster quality. As shown in this chapter and the work on which

² In nine cases the majority vote resulted in a draw between two equally frequent classes. We resolved this by assigning the class of the instance most similar to our test instance. For this nearest-neighbor voting only instances of the cluster belonging to one of the two equally frequent classes were taken into account.

it is based, it is possible to define useful distance measures for first-order objects, making the strengths of distance-based methods available to first-order problems. Whenever an explicit generalization is not needed, the methods presented here are very strong contenders in terms of solution quality.

However, a few words of caution are also in order with respect to scalability. Firstly, instance-based learning methods must store all (or, when optimized variants are used, at least some) of their instances until classification time. Thus, the compression effect that can be obtained with explicit hypotheses is not achieved here. Secondly, computing distances between first-order objects is not as cheap an operation as computing a Euclidean distance (see [9.8] for details). While still faster than most generalization algorithms, this distance computation must be carried out at classification time, so such methods are not suitable if very rapid classification is a primary concern. As to the clustering methods, once the distance matrix is computed, they largely inherit the known computational properties of standard distance-based clustering algorithms; however, convergence of k -means may be somewhat slower since no exact cluster centers are computed.

In sum, all three of the methods described here are most suitable for domains with no more than several thousands of objects; more research is needed to scale them to the very large databases sometimes found in KDD.

References

- 9.1 G. Bisson. Conceptual Clustering in a First-Order Logic Representation. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 458–462. John Wiley and Sons, Chichester, 1992.
- 9.2 U. Böhnebeck, T. Horváth, and S. Wrobel. Term comparisons in first-order similarity measures. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 65–79. Springer, Berlin, 1998.
- 9.3 W. Dillon and M. Goldstein. *Multivariate analysis*, pages 157–208. John Wiley and Sons, Chichester, 1984.
- 9.4 S. Džeroski, S. Schulze-Kremer, K. Heidtke, K. Siems, and D. Wettschereck. Diterpene structure elucidation from ^{13}C NMR spectra with machine learning. In N. Lavrač, E. Keravnou, and B. Zupan, editors, *Intelligent Data Analysis in Medicine and Pharmacology*, pages 207–225. Kluwer, Boston, 1997.
- 9.5 W. Emde and D. Wettschereck. Relational Instance-Based Learning. In *Proceedings of the Thirteen International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, San Francisco, CA, 1996.
- 9.6 J. Hartigan. *Clustering Algorithms*, pages 58–73. John Wiley and Sons, Chichester, 1975.
- 9.7 T. Horváth, Z. Alexin, T. Gyimóthy, and S. Wrobel. Application of different learning methods to Hungarian part-of-speech tagging. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, pages 128–139. Springer, Berlin, 1999.
- 9.8 T. Horváth, S. Wrobel, and U. Böhnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2): 53–80, 2001.

- 9.9 A. Hutchinson. Metrics on Terms and Clauses. In *Proceedings of the Ninth European Conference on Machine Learning*, pages 138–145. Springer, Berlin, 1997.
- 9.10 L. Kaufmann and P. J. Rousseeuw. Clustering by means of medoids. In Y. Dodge, editor, *Statistical Data Analysis based on the L_1 Norm*, pages 405–416. Elsevier, Amsterdam, 1987.
- 9.11 L. Kaufmann and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley and Sons, Chichester, 1990.
- 9.12 M. Kirsten and S. Wrobel. Relational Distance-Based Clustering. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 261–270. Springer, Berlin, 1998.
- 9.13 M. Kirsten and S. Wrobel. Extending k -means clustering to first-order representations. In *Proceedings of the Tenth International Conference on Inductive Logic Programming*, pages 112–129. Springer, Berlin, 2000.
- 9.14 P. Mahalanobis. On the generalized distance in statistics. *Proceedings of the Indian National Institute Science*, 2: 49–55, Calcutta, 1936.
- 9.15 J. McQueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–293, 1967.
- 9.16 S.-H. Nienhuys-Cheng. Distance Between Herbrand Interpretations: A Measure for Approximations to a Target Concept. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 213–226. Springer, Berlin, 1997.
- 9.17 S.-H. Nienhuys-Cheng. Distances and limits on Herbrand interpretations. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 250–260. Springer, Berlin, 1998.
- 9.18 J. Ramon and M. Bruynooghe. A framework for defining distances between first-order logic objects. *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 271–280. Springer, Berlin, 1998.
- 9.19 M. Sebag. Distance Induction in First Order Logic. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 264–272. Springer, Berlin, 1997.
- 9.20 B. A. Shapiro and K. Zhang. Comparing Multiple RNA Secondary Structures Using Tree Comparisons. *Computer Applications in Biosciences*, 6(4): 309–318, 1990.
- 9.21 A. Srinivasan, S. Muggleton, and R. D. King. Comparing the use of background knowledge by inductive logic programming systems. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming*, pages 199–230. Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 1995.
- 9.22 E. Ukkonen. Algorithms for Approximate String Matching. *Information and Control*, 64: 100–118, 1985.
- 9.23 D. Wettschereck and D. Aha. Weighting Features. In *Proceedings of the First International Conference on Case-Based Reasoning*, pages 347–358. Springer, Berlin, 1995.
- 9.24 K. Zhang and D. Shasha. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *SIAM Journal on Computing*, 18(6): 1245–1262, 1989.
- 9.25 M. Zuker and P. Stiegler. Optimal Computer Folding of Large RNA Sequences Using Thermodynamics and Auxiliary Information. *Nucleic Acids Research*, 9(1): 133–148, 1981.