Cover Page

Universiteit Leiden

Author: Witsenburg, Tijn
Title: Hybrid similarities : a method to insert relational information into existing data mining tools
Date: 2012-12-20

Hybrid Similarities

a method to insert relational information
into existing data mining tools

Tijn Witsenburg

Hybrid Similarities

a method to insert relational information into existing data mining tools

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van de Rector Magnificus prof.mr. P.F. van der Heijden,
volgens besluit van het College voor Promoties
te verdedigen op donderdag 20 december 2012
klokke 16:15 uur

door

Martinus Damianus Jacobus Witsenburg
geboren te Leiden
in 1975

Promotiecommissie

Promotor:	Prof.dr. J.N. Kok	
Promotor:	Prof.dr. H. Blockeel	(Katholieke Universiteit Leuven)
Overige leden:	Dr. W.A. Kosters	
	Dr. P. van der Putten	
	Prof.dr. A.P.J.M. Siebes	(Universiteit Utrecht)
	Prof.dr. T.H.W. Bäck	

This research was financed by the Netherlands Organisation for Scientific Research (NWO) under project 639.022.605 “Annotated Graph Mining”.

Druk: *Mostert en Van Onderen!*

Contents

Chapter 1

Introduction

1.1 Motivation

Little could the creators of the first computers suspect the impact they would have on society nowadays. Invented as machines to help men with difficult calculations, they soon evolved into convenient tools for storing and sharing huge quantities of information. Statisticians Lyman and Varian suggested in 2003 that worldwide data volumes will double every two to three years [60]. These data can be anything like the results of scientific research, internet, customer behaviour, video's, computer games, and so on. Besides the amount of data that is stored, also the amount of data that is consumed can give somewhat of a perspective to the extend to which information influences our behaviour. Bohn and Short stated that the amount of data consumed by Americans in 2008 was 3.6 zettabytes, which is $3.6 \cdot 10^{21}$ bytes, or $1,000,000,000,000$ gigabytes [6].

The collecting and storing of data has an increasing role in society where everyday activities are more and more being digitalised. Examples are: people using social network sites create a network of people, super markets that use 'customer cards' to monitor customer behaviour, public transport companies that use electronic checking in and out, and governments that stimulate people to fill in their tax form electronically.

All this sort of behaviour creates huge databases, containing potentially interesting information for sociologists, marketers, demographists, and various different types of scientists. While this enormous amount of data gives scientists many opportunities to conduct research, it is difficult for them to actually extract the useful information. Data nowadays can consist of billions of records, or observations, each having many different variables (also known as attributes). The large amount of attributes, or high-dimensionality of the data, makes it impractical to compare each pair of attributes. Also, the large amount of records limits the amount of times that an algorithm can pass through the whole data

set. Classical statistical analysis methods are not capable of handling such amounts of data. Therefore a new type of methods needs to be invented.

The field of research that addresses this problem is known as Knowledge Discovery in Data (KDD). The most important step in the KDD process is known as data mining. Data mining is concerned with "applying computational techniques (i.e. data mining algorithms implemented as computer programs) to actually find patterns in the data", as defined by Džeroski [20]. These 'patterns' that Džeroski describes are not explicitly defined and depend on both the task that the data mining tool needs to perform, and the type of data it is performed on.

A data mining task can be seen as "what the user wants to know". For instance, if the user wants to predict a label for an element in the data set, then the data mining tool should be designed to do exactly that. Other examples of data mining tasks can be to divide the elements into groups of similar type, or to find combinations that occur often. In each case, of course, the pattern to be found is defined in a different way. On a more abstract level, these data mining tasks all search for the same: some underlying structure in the data that gives rise to patterns between elements.

The patterns to be found also depend on the type of data. Earlier data mining tools work only on data that is in a single table. Here, every row is considered one object of interest and every column an aspect, or attribute, of that object. Unfortunately, data do not always come in the same shape. Though there are many databases that consist of data in the form of a single table, many more do not. Examples of different types of data are relational data, graph data, video data, time sequence data, et cetera. Since data mining tasks in general only work on one type of data, for every other type, new data mining tools need to be invented.

To make things more complicated, a database could consist of a combination of more than one type of data. A very popular example nowadays are social networks, where all information about the people in it (e.g. their interests and likes) could be placed in a single table, but the friend structure is considered relational data. A second example would be a data set with scientific papers. Here the content (words that appear in the text) can be placed in a single table, and there is also relational information in the form of citations between papers, or papers having an author in common.

Data mining tools tend to use only one type of data. When such a data mining tool is applied to a database with multiple types of data, it cannot use all the information available in this database. In theory, this does not necessarily need to be a problem. The solutions that are found with data mining tools that use only part of the data can be just as correct, as solutions found with a data mining tool that uses all the information in the database. It could even be so that there is just as much confidence in the quality of the found solutions. Despite all that, in practice, a data mining tool that is able to use all the

information in the database, could explore a much wider search space and would therefore, in potential, be a much more powerful data mining tool.

In this thesis, a data mining tool that uses more than one type of data is considered a hybrid data mining tool. When a hybrid data mining tool is created for a specific combination of data types, it will probably be very powerful on a database that has that combination of data types, but not perform well on another database that consists of a different combination of data types, should it even be able to use this other database as input.

There are many different types of data. The amount of different types of databases that can be created from combining multiple types of data, is exponential in relation to the amount of data types and thus very big. It is an incredibly elaborate task to create a new hybrid data mining tool for every possible combination of data types and every data mining task. So, in order to create hybrid data mining tools, a more general approach is needed. One such approach could be to create a method that incorporates one type of data in a data mining tool which is created for another type of data.

In this thesis, we propose a method that does this by inserting relational information in data mining tools that are created for single table data and use a similarity measure to compare the elements in the database with each other. It can also be used for other types of data, as long as the data mining tool uses a similarity measure. This leaves a wide range of possible data mining tools that can be enhanced by using this method. For various data mining tools, it has been implemented and tested. Although the results varied, overall, this method outperformed the original data mining tools.

1.2 Thesis Outline

This thesis consists of four parts: Foundations, Implementations, Analysis and Conclusions. Part I *Foundations* starts in Chapter 2 where the current state of data mining is described. It concentrates on the aspects that are important for this thesis. Then in Chapter 3 the method proposed in this thesis is explained. This is done by firstly describing the kind of data for which this method is created, and secondly, precisely describing the method itself. For the latter, two different approaches are used: one graph-based approach and one matrix-based approach. Also in this chapter, the method is compared to other methods that can combine different types of information. The final chapter in this part, Chapter 4, gives an intuitive motivation on why the proposed method could work better than already existing data mining tools and from this extracts some characteristics of the data for which this method could be useful.

The method proposed in this thesis can be used to improve already existing data mining tools. Part II *Implementations* describes the data mining tools for which this is done and gives the results on real data for the improved methods in comparison to the original data mining tools. Chapter 5 does this

for agglomerative hierarchical clustering, Chapter 6 does this for k-means and Chapter 7 does this for KNN-classification.

Part III *Analysis* takes a closer look at the exact working of the method. First, in Chapter 8 the mathematical benefits of the proposed method are analyzed. Then, in Chapter 9 alternative possibilities for the proposed method are constructed and compared them with each other.

Finally, Part IV *Conclusions* gives an overview of all the conclusions that can be drawn from this thesis.

1.3 Publications

Several of the chapters in this thesis are based on publications. Listed below are the publications for each chapter.

Chapter 4: Motivation

This chapter is based on a paper presented at MPS'10:

> Hendrik Blockeel and Tijn Witsenburg. Exploiting homophily in unsupervised learning. In *Workshop on Mining Patterns and Subgroups*, Leiden, The Netherlands, 2010.

Chapter 5: Agglomerative Hierarchical Clustering

This chapter is based on a paper presented at MLG'08:

> Tijn Witsenburg and Hendrik Blockeel. A method to extend existing document clustering procedures in order to include relational information. In S. Kaski, S. Vishwanathan, and S. Wrobel, editors, *Proceedings of the 6th International Workshop on Mining and Learning with Graphs*, Helsinki, Finland, 2008.

Chapter 6: K-means and K-medoids

This chapter is based on a paper presented at ISMIS'11:

> Tijn Witsenburg and Hendrik Blockeel. K-means based approaches to clustering nodes in annotated graphs. In M. Kryszkiewicz, H. Rybinski, A. Skowron, and Z.W. Ras, editors, *Foundations of Intelligent Systems, ISMIS'11 Proceedings*, pages 346–357, Warsaw, Poland, 2011.

Chapter 8: Reducing the Impact of Content Variability

This chapter is based on a paper presented at the 19th International Symposium on Methodologies for Intelligent Systems:

Tijn Witsenburg and Hendrik Blockeel. Improving the accuracy of similarity measures by using link information. In M. Kryszkiewicz, H. Rybinski, A. Skowron, and Z.W. Ras, editors, *Foundations of Intelligent Systems, ISMIS'11 Proceedings*, pages 501–512, Warsaw, Poland, 2011.

Other Publications

The following publications on related subjects were co-authored during the PhD thesis:

Hendrik Blockeel, Tijn Witsenburg and Joost N. Kok. Graphs, Hypergraphs and Inductive Logic Programming. In P. Frasconi, K. Kersting and K. Tsuda, editors, *Proceedings of the 5th International Workshop on Mining and Learning with Graphs*, pages 93–96, Firenze, Italy, 2007.

Hendrik Blockeel, Hossein Rahmani and Tijn Witsenburg. On the importance of similarity measures for planning to learn. *Proceedings of the 3rd International Workshop on Planning to Learn*, Lisbon, Portugal, 2010.

Part I

Foundations

Chapter 2

Background

Data mining algorithms aim to retrieve useful pieces of information from huge data sets. A very important aspect in the creation of successful data mining algorithms is a good understanding of the type of data that it needs to work on. We distinguish between two main categories: content-based data and relational data. A fairly new field of research consists of data mining algorithms that are designed for hybrid data, i.e., datasets that consists of both types of data.

2.1 Introduction

In the last decades, the amount of data has grown exponentially, resulting in more data sets of increasing size. Data nowadays can consist of billions of records, or observations, each having many different variables, or attributes. The large amount of attributes, or high-dimensionality of the data, makes it impractical to compare each pair of attributes. Also the sheer amount of records limits the number of times that an algorithm can pass through the whole data set.

This calls for a whole new approach to analysing data, called *Knowledge Discovery in Databases* (KDD). It has been defined by Fayyad et al. as the "non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data" [23]. This definition is very broad, and whether a found pattern is valid, novel, useful, and understandable, is in the eye of the beholder; in this case the user. KDD knows three important steps: prepare the raw data to a usable form, find interesting patterns, and evaluate and interpret the found patterns.

This chapter aims to give an overview of this process in which it focusses on the difficulties that occur when the data is presented in a different format. It is organized as follows: Section 2.2 gives an overview of the most important step in the KDD process. Section 2.3 gives an overview of different types of data

and explains why they are essentially different and what the effect on the KDD process is. Section 2.4 explains what hybrid data is (the type of data where the method of this thesis focusses on). Finally, Section 2.5 gives an overview of other methods that also aim to work on hybrid data.

This chapter is loosely based on *Introduction to Data Mining* by Tan, Steinbach and Kumar [94], *Relational Data Mining* by Džeroski [20], and *Graph-Based Data Mining* by Cook and Holder [17].

2.2 Data Mining

The step in the KDD process that searches for patterns can be seen as the central step and is known as *data mining*. It has been defined more precisely by Džeroski as "a step in the KDD process concerned with applying computational techniques (i.e. data mining algorithms implemented as computer programs) to actually find patterns in the data" [20]. The crucial role of the data mining step in the KDD process is underlined by the fact that sometimes the whole KDD process is refered to as 'data mining.' Though this introduces some ambiguity, it hardly ever leads to confusion.

Many data mining algorithms come from the field of *machine learning*, which is defined by Russell and Norvig as the capability of a program to "adapt to new circumstances and to detect and extrapolate patterns" [83]. The ability to 'adapt to new circumstances' is considered learning, hence the name 'machine learning.' This is very useful since it allows for scalability; the algorithm can start with part of the data and adapt as more data is reviewed. It also enables the algorithm to learn from data that change over time (for instance, because it is 'live' data where new events are added at the moment they occur).

The patterns that are extracted do not have a predefined form. They can be anything, depending on the type of data and which data mining task is done. The first data mining tools work on data that is in the single table form. Here, every row is an element, or observation, and every column is one of its attributes. In this section, it is assumed that the data is in this form, but then still, the extracted patterns depend on the task. Most of the time, the result of a data mining task is not literally a pattern that can be found in the data. Instead, the result describes the effect of an underlying pattern on a more abstract level.

There are different data mining tasks and there are different ways to qualify them. From machine learning, there is the distinction between supervised and unsupervised learning. In *supervised learning* the algorithm is provided with pairs of data, consisting of input elements with the desired output. The algorithm then needs to learn a function which can predict the output for elements that are not in the training set. This is in contrast to *unsupervised learning* where the algorithm is provided with a set of data elements that do not have a known desired output. In that case, the algorithm tries to find the struc-

ture of the input data. Between supervised and unsupervised learning, there is *semi-supervised learning*, where only part of the desired output is known.

Another way to qualify data mining tasks is the distinction between predictive and descriptive learning. In *predictive learning*, the algorithm tries to predict a certain value, for instance that of an attribute or label of an element. Most of the time, there is a data set where the value that needs to be predicted is already known, and thus can be used to train the algorithm. Since this is known as supervised learning, it is not surprising that many tasks that are considered predictive learning are also considered supervised learning. In *descriptive learning*, the algorithm tries to describe the data. This is mostly done by finding some underlying structure, the latter being known as unsupervised learning. Indeed, many data mining tasks that are known to be descriptive learning are also known to be unsupervised learning.

One of the most important examples of a data mining task that is both supervised and predictive learning, is classification. This is explained more elaborately in Subsection 2.2.1. Subsection 2.2.2 describes clustering, which is a very important data mining task that is both unsupervised and descriptive learning. To conclude this section, Subsection 2.2.3 describes some other data mining tasks.

2.2.1 Classification

An important example of a data mining task is *classification*. Here, the algorithm needs to assign one of a set of predefined classes to an object. These classes are discrete values from a finite set. Within machine learning it is an example of supervised learning. Within data mining, classification is known as predictive learning.

A typical example is a data set filled with the characteristics of vertebrates where attributes can be 'Skin Cover', 'Gives Birth', 'Aquatic', and so on. For each species in the data set the class ('amphibian', 'bird', 'fish', 'mammal', or 'reptile') is given as well. From this input data, a classification model can be constructed that predicts a class for any new creature, once the values for its attributes are known.

Classification is not only used in taxonomy. There are many application domains where classification can be applied. Examples are spam detection (compare the content of an e-mail to e-mails that are known to be spam), predicting a protein's involvement in the development of cancer, and classifying galaxies based upon radiation patterns. Furthermore, classification is also used as part of other techniques. For instance, pattern recognition techniques use classification as part of their process. Examples are speech recognition, image classification and identifying characters that are handwritten.

Generally, the algorithm uses a subset of the dataset to create a function that gets the values of the attributes of an element as input and then returns a

prediction for the class as output. This subset is called the training set. When a function is found, a validation set is used to check for overfitting. Overfitting is when the function is specialized to classify the training set, but performs poorly on elements outside this set. When this is done, the function is used to predict the classes for the rest of the elements in the dataset, which is then called the test set. This gives an estimation of how well the function predicts when a new, yet unknown, element is presented to the function.

One of the first works on classification was by Fisher [24] who found a way to determine to which of two classes a new element would most likely belong, using his *lineair discriminant function* and assuming that data values within each of the two groups had a multivariate normal distribution. Later, Rao [81] improved the method to more than two groups, provided the classification rule was linear (a restriction that Anderson [2] made redundant).

Besides functions, another form of classifiers are *decision trees*. Here, the result is a tree describing a series of questions about the attributes. Starting in the root of the tree, whilst evaluating an object, different answers to a question will lead to different subtrees and thus different follow-up questions. This process is repeated until a leaf node is reached. The class label associated with this leaf node is the class that is assigned to the object. Examples of well-known decision tree algorithms are CART by Breiman et al. [9], ID3 and C4.5 by Quinlan [75, 77] and CHAID by Kass [44].

A third form of classifiers are the *rule-based classifiers*. Here the resulting classifier will be a set of "if ... then ..." rules. The expressive power of rule-based qualifiers is greater than that of decision trees. This can be seen easily when one notices that every path in a decision tree can be translated to a rule, but not every set of rules can be translated to a decision tree. This makes it easier for rule-based classifiers to find a good classifying model for a certain data set. The downside is that it can be possible that the resulting set of rules are inconclusive or conflicting. Important rule-based classifiers are 1R by Holte [36] or RIPPER by Cohen [13].

Other classifying methods are *nearest neighbor classifiers* (e.g. Cover and Hart [18] or Han et al. [31]), *naive Bayes classifiers* (e.g. Langley et al. [54]), *Bayesian belief networks* (e.g. Heckerman [34]), or *support vector machines* (see Vapnik [96, 97], or Shawe-Taylor and Cristianini [84]). Of these methods, the nearest neighbor classifiers are of particular interest for this work. They are discussed in more detail in Chapter 7.

2.2.2 Clustering

Another important example of a data mining task is clustering. *Clustering* is a technique where the data set is divided into subsets, called clusters, of elements that belong to each other. Within machine learning, it is known as unsupervised learning, and within data mining it is known as descriptive learning. Normally,

clustering algorithms use a function that gets the attributes of two elements as input and returns a similarity measure that defines how similar these two elements are. The clustering algorithm then tries to form clusters where all elements in the same cluster are as similar as possible while elements that are not in the same cluster are as dissimilar as possible.

There are two main tactics used by clustering algorithms. The first is hierarchical, where new clusters are derived from existing ones. This can be done bottom-up or top-down. Bottom-up hierarchical clustering is called agglomerative hierarchical clustering. Here, the algorithm starts with clusters of one element that are iteratively merged to form increasingly bigger clusters. Top-down hierarchical clustering is called divisive hierarchical clustering. Here, all elements start in one cluster that is split up repeatedly to come up with smaller clusters. The result of a hierarchical clustering method is most of the times presented in a dendrogram, which completely shows the found hierarchy. More about agglomerative hierarchical clustering can be found in the book of Sneath and Sokal [87]. An example of divisive hierarchical clustering is PDDP by Boley [8].

The second main tactics for clustering algorithms is where the algorithm initially creates a certain number of clusters that are then iteratively adjusted by moving elements from one cluster to another until a stopping criterium is met. The best known clustering algorithm that uses this principle is k-means by MacQueen [61].

There are clustering algorithms that use a completely different approach nonetheless. One example is DBSCAN by Ester et al. [22]. Here the algorithm starts by considering all elements unclustered. For an arbitrary, unclustered element, it checks whether in its vicinity (all distances smaller than a predefined ϵ) are at least a minimum amount (predefined by $minPts$) of elements. Said otherwise, it checks whether the vicinity of an element is dense enough. Then, at this point starts a new cluster that is expanded with all elements that are within the range of ϵ and are in a vicinity that should have been dense enough to start a cluster. After this cluster is formed, the whole process is repeated with a new, arbitrary, unclustered element. This procedure continues until no more clusters can be added.

2.2.3 Other Data Mining Tasks

There are also other data mining tasks. Some of them will be discussed in this section.

With *association analysis*, the aim is to discover interesting relationships between elements in the data set. Within machine learning, it is an example of unsupervised learning. The algorithm does not get any examples of what those interesting patterns could be. An important type of association analysis is association rule mining which mainly handles market basket transactions.

Market basket transactions are data where each row is a transaction made by a customer consisting of all the items bought by that customer with that transaction. When applying association rule mining to this type of data, relationships between elements are considered interesting when they often occur simultaneously in a transaction. Association rule mining was first introduced by Agrawal et al. with their APRIORI algorithm [1]. Other examples are DHP by Park et al. [74], DIC by Brin et al. [10], and FP-GROWTH by Han et al. [33].

Classification predicts a class for an element, where this class is one value from a discrete set of values. When the value that needs to be predicted is a continuous value, the task is known as *regression*. While these tasks are in principle the same (i.e. predicting the value for an element in the data set) their approaches are completely different, due to the different nature of the value that needs to be predicted. Therefore, classification and regression are considered to be two different data mining tasks. Regression is used to predict, for instance, the value for a stock after a certain time, the total sales of a company, the amount of rain that will fall in a certain region and during some season, and so on.

With *concept learning*, the task is to find a model that can predict whether an element belongs to a certain class or not. It is therefore also known as binary classification, which classifies whether it is TRUE or FALSE whether an element belongs to a certain class. It is used to predict, for instance, whether a cell is cancerous, whether someone should be allowed to get a loan, whether some protein could be useful for inventing a new medicine, and so on. Concept learning differs from classification, though, in that in concept learning, the model itself is typically considered the goal, not the ability to make predictions. The model should be an interpretable description of the concept that has been learned.

2.3 Data Types for Data Mining

An important aspect in the design of a data mining algorith is the type of data that is used. The early data mining algorithms were all designed for flat data, that is, data that only consists of elements with their attributes. This means that the data can be seen as a single table where the rows are the elements, or observations, and the columns are the attributes. Not all data can be described as one single table, creating a problem for data mining tools that can only work on this type of data. This problem can be solved in two different ways: either by transforming the data so it fits in a single table, or by altering the data mining tools in such way that they can handle different types of data.

Transforming the data to a single table can be very difficult. Consider, for instance, the database of a company that sells products to customers. Table 2.1 gives an example of what a table with all the customer information would look like. Now, if there is other general information about these customers (like, for

instance, their address, income, the amount of children they have, and so on) and the company want it included in the database, then this can easily be done by adding another column for each attribute that the company is interested in.

ID	Name	Sex	Location
C0001	Mies	F	Leiden
C0002	Wim	M	Leiden
⋮	⋮	⋮	⋮
C0119	Jet	F	Rotterdam

Table 2.1: Example of a table with customer information

Besides information about their customers, the company also has a table with information about their products. Table 2.2 gives an example of how such a table could look like. Since customers and products are two completely different types of entities, two different tables are created: one with customer information and one with product information.

ID	Name	Type	Price
P1575	Gaffer Tape	attacher	42
P1908	Mobitronic	regulator	1234
⋮	⋮	⋮	⋮
P2012	Tie-Wraps	attacher	1

Table 2.2: Example of a table with product information

If the company wants to use a data mining tool that works on data that is in the single table form, it has to choose one of the two tables. Depending on whether the company wants to know more about their customers or their products, it takes the appropriate table. Since there is no relation between the customers and the products, this should not be a problem.

Things get a bit more complicated, when the company decides to keep track of the purchases of their customers. They would like to know who buys what item, how many of them and on what date. This information can not be included in either the customer table or the product table. Customers can buy different products and products can be sold to different customers, and all this can be done for different amounts and on different dates. This makes it impossible to include this information as a set of new attributes in any of the two already existing tables.

Normally, in such a case, this company would create a new table in which information about purchases is stored. Table 2.3 is an example of how such a table would look like. From this table we can see that on February 8, Jet

bought one hundred tie-wraps and five rolls of gaffer tape, and on July 19, Wim bought one mobitronic.

Customer	Product	Date	Amount
C0119	P2012	February 8	100
C0119	P1575	February 8	5
⋮	⋮	⋮	⋮
C0002	P1908	July 19	1

Table 2.3: Example of a table with information about purchases

Information about purchases is very interesting for a company, and it can easily be seen how a company would like to use this information in data mining tools. For instance, the company would like to group their customers to see what kind of customers they have, which could be nice to know when a new advertisement campaign is to be launched. Another example could be that the company would like to classify the customers so that people who are classified in the category 'good customers' could get some sort of discount.

For these two examples, information from multiple tables is needed. When grouping the customers, both their behaviour and their personal information could be used. To see if someone could be considered a good customer, it is not only interesting to know how many products they bought (which is in the purchases table), but also what the price of these products is (which is in the products table). Thus, if the data mining tool we are using only can work on data that is in the single table form, and it is insufficient to use only one of the three tables, a solution must be found to put all data into a single table.

One option could be to take the purchase table, expand it with the attributes of customers and products and for every tuple fill in the appropriate information. In relational database terminology, this is called an INNER JOIN. The problem is that now a lot of information will appear multiple times in this table. This could influence the data mining tool and makes it more difficult to keep the database accurate.

Another option could be to summarise customer behaviour, but this means serious loss of information. Even other ways to combine all information in a single table could be constructed, but all have their downsides, most of the time resulting in loss of information or information appearing multiple times. All in all, databases like this are very difficult to be put in single table form.

For other types of data, it will be even more difficult to put them in a single table form. Consider data sets that consist of pictures, molecules, texts, videos, or family trees. It is impossible to translate any of these examples to a single tuple of attribute values, without serious loss of information. Therefore, it is more interesting to try to alter existing data mining tools, or develop completely

new ones, that are capable of performing data mining tasks on other types of data.

Just as the data mining tools described in Section 2.2 only perform well on data in a specific form, the same holds for data mining tools that work on other types of data. Much research has been done on data mining tools for relational databases. This is called relational data mining, which is described in Section 2.3.1. Another form of data mining that received a lot of attention is graph data mining, which, obviously, researches data mining tools for graph data. It is described in Section 2.3.2.

With relational data mining and graph data mining, the two major groups of data mining tools that do not use a single table have been named. Other types of data, for instance pictures or videos, ask for even different types of data mining tools. Most of the time, in the development of these tools, the big issue is in the data-preparation phase, where much domain knowledge is necessary. Since these are very specific techniques, and any knowledge about them is not needed for the understanding of this thesis, they will not be discussed here.

The term data mining often refers to data mining on single table data. Nonetheless, it is also often used as a collective term for all data mining tools, including relational data mining, graph data mining and all other data mining tools on data that is not in the single table form. To make things more clear, in this thesis the term *standard data mining* is used whenever data mining on data in the single table form (and not data mining on other types of data), is meant.

2.3.1 Relational Data Mining

Many datasets nowadays are relational databases. Originally defined by Codd in 1970 [12], the relational database framework made it possible to create a database with many different types of elements that can have many different types of relations between them. It is a very powerful tool, that enables creators of databases to design very complex databases. More information on relational databases can be found in the book of Connolly and Begg [15]. With regards to data mining, these type of databases have their downside. Data mining tools that are created for single table data are not suited to perform their tasks on relational databases.

A possible solution to this problem came from the field of *inductive logic programming* (ILP). ILP mainly uses languages based on logic programming. Logic programming is based upon first-order logic, also known as predicate logic. Relational databases are formally created with relational algebra, which is also based upon first-order logic. This can easily be seen when it is noticed that the predicates correspond to the relations and that the arguments of a predicate correspond to the attributes of a relation. More information about first-order logic can be found in the book of Reeves and Clarke [82].

ILP was first concerned with the creation of logic programs from examples and background knowledge. This task is also known as relational rule induction and can be seen as concept learning. Some of the early ILP algorithms are CIGOL by Muggleton and Buntine [71], and FOIL by Quinlan [76]. More recent examples include PROGOL by Muggleton [70] and ALEPH by Srinivasan [90].

When it became more and more clear that ILP is a very powerful tool for learning concepts in relational data, scientists started exploring the ability of ILP to perform other data mining tasks. This resulted in the expansion of the possibilities of ILP, which now can perform all the major data mining tasks: classification, clustering and association analysis. Many of them are improvements of already existing standard data mining algorithms, where the possibilities created by ILP allow for the inclusion of relational information.

Much attention has gone to relational classification where already many algorithms have been developed. Examples are: Inductive Classification Logic (ICL) by De Raedt and Van Laer [80], an altering of CN2, Structural Classification and Regression Trees (S-CART) by Kramer [50, 51], an altering of CART, Top-down Induction of Logical Decision Trees (TILDE) by Blockeel and De Raedt [4, 5], an altering of C4.5, and RIBL2 by Bohnebeck et al. [7], an adaptation of KNN-classification.

Also relational clustering algorithms and relational association analysis algorithms have been developed. Examples of relational clustering are: RDBC by Kirsten and Wrobel [47], an altering of agglomerative hierarchical clustering, and FORC by the same authors [48], an altering of k-means. Examples of relational association analysis are CLAUDIEN by De Raedt and Bruynooghe [79], WARMR by Dehaspe and Toivonen [19].

2.3.2 Graph Data Mining

Relational data mining tools are designed to work on relational databases, created with relational algebra. Another well-known type of relational data is graph data. A graph is a set of vertices with edges connecting them. From this area of computer science, a whole new range of data mining tools is developed. Graphs are, just as relational databases, a way to formalize relations between elements.

A graph can easily be turned into a relational database. Every (type of) vertex corresponds to an entity, and every (type of) edge corresponds to a relation between two entities. Any information like labels, numerical values, and so on, that is attached to a vertex or edge corresponds to an attribute of that entity or relation. For example, the graph in Figure 2.1 can be transposed to the relational database described in Table 2.4.

The graph has two types of vertices, resulting in two different tables for the entities 'square' and 'circle'. From this, three types of relations can be described: edges that connect two squares, edges that connect two circles, and

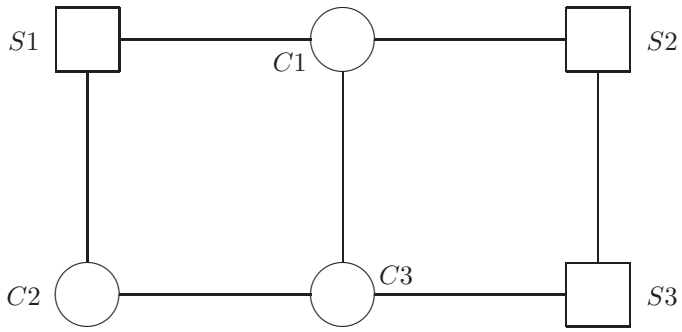


Figure 2.1: Example of a graph with two different kinds of vertices.

Squares

ID
$S1$
$S2$
$S3$

Circles

ID
$C1$
$C2$
$C3$

Square-to-Square

Square1	Square2
$S2$	$S3$

Circle-to-Circle

Circle1	Circle2
$C1$	$C3$
$C2$	$C3$

Square-to-Circle

Square	Circle
$S1$	$C1$
$S1$	$C2$
$S2$	$C1$
$S3$	$C3$

Table 2.4: Relational database describing the graph in Figure 2.1.

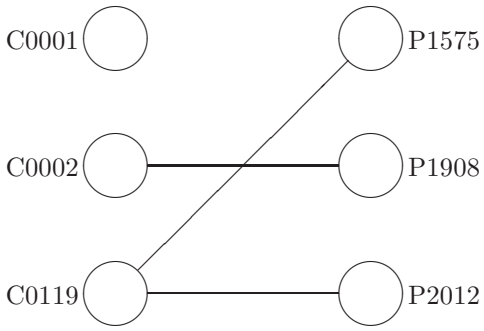


Figure 2.2: The structure of the relational database described in Tables 2.1 through 2.3 drawn as a graph.

edges that connect a square to a circle. Any additional information about vertices or edges (labels or annotations) can easily be included as an attribute in the corresponding table.

Transposing a relational database to a graph is not so straightforward. Considering the database from Tables 2.1 through 2.3, and considering only the tuples shown in the example, the structure of this database could be described in a graph like the one drawn in Figure 2.2. Here, the customers are on the left and the products on the right. Purchases can be seen as the edges, combining a customer with a product.

The attributes of the tables can be included in different ways. One is simply to include them all in the label of the element. The label 'C0001' would then be 'C0001;Mies;F;Leiden' and the edge between 'C0119' and 'P1575' would be 'February 8;5'. The downside of this, is that it is difficult to see when two fields are the same. For instance, Mies and Wim both come from Leiden. This could be interesting information for a clustering algorithm, but it is lost when it is included in the label since the labels 'C0001;Mies;F;Leiden' and 'C0002;Wim;M;Leiden' are different.

An alternative for putting the attribute values in the labels would be to create a vertex for each attribute value in the database and connect values that belong to each other. An example of how to do this for Table 2.1 is drawn in Figure 2.3. For clarity reasons, only the customers table is represented. It can be done for the other tables as well, but this would create a graph that is so large that it becomes unclear what the relation is with the original tables.

In this case it is much clearer to see when customers have attributes in common. For instance, customer 'C0001' and 'C0002' are both connected to 'Leiden', and thus they live in the same city. The problem is that it is much harder to see the more general structure of the database, the one described in Figure 2.2. Despite the fact that both methods have their own disadvantages,

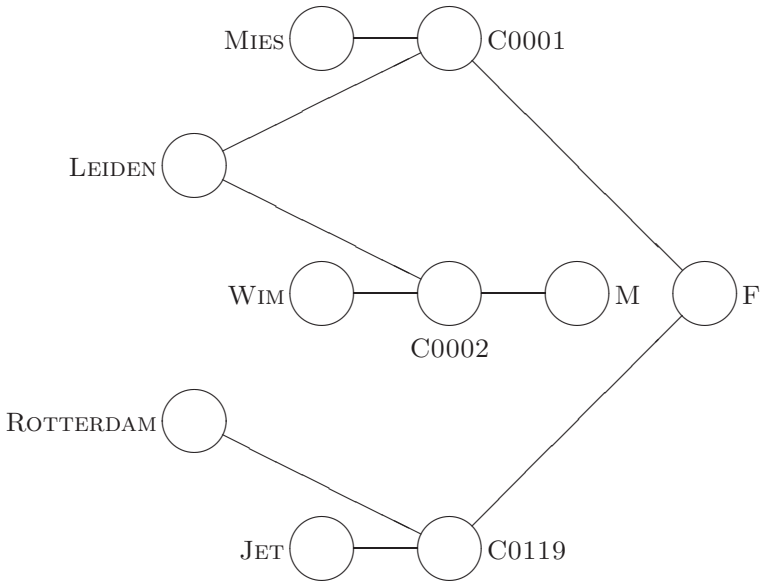


Figure 2.3: The relational database described in Table 2.1 drawn as a graph where each attribute value is a vertex.

it is possible to translate a relational database to a graph and keep the same information available. This shows that relational databases and graphs are different ways to describe the same thing: data that has relational information.

Although both data types describe the same kind of data, due to their different representation, they have given rise to completely different kinds of data mining tools. Relational data mining tools are created for data that is in the shape of relational databases, and thus they are often based on first-order logic. Graph mining tools are created for data that is in the shape of graphs, and thus they are often based on linear algebra and graph manipulation actions.

In graph data, two main groups can be distinguished: graph data that consists of one big graph, or graph data that consists of a set of graphs. When the data set is one big graph, the main objects for the data mining tool are the vertices in the graph (and in some cases the edges). In this case, graph data mining tools try to classify the vertices, or construct clusters that are subsets of vertices. When the data set is a set of graphs, the main objects for data mining are these graphs. Here, classification means classifying one of the graphs and clustering means grouping the graphs into subsets.

In graph data mining, once again, the three main data mining tasks can be distinguished: classification, clustering and association analysis. Classification for graph mining can have different interpretations. When the data set consists

of a set of graphs, these graphs needs to be classified. Kernel methods to do so have been developed by Gärtner et al. [27] and Horváth et al. [38]. When the data consists of one big graph, the vertices in this graph need to be classified. A kernel method for classifying vertices has been developed by Kondor and Lafferty [49].

In graph clustering, mostly unlabeled graphs are considered. Here clustering means finding subgraphs where the vertices within a subgraph are highly connected whilst the amount of connections between subgraphs is as low as possible. A somewhat more advanced setting is obtained when the edges in the graph are weighted (i.e. have numerical labels). In this case it is most common to try to find subgraphs where the sum of the weights of all edges within a subgraph is as high as possible while the sum of the weights of all edges between different subgraphs is as low as possible. Examples of graph clustering methods are SAHN by Sneath and Sokal [87], PAM by Kaufman and Rousseeuw [46], the Girvan-Newman algorithm by the same authors [29], Karger's MIN-CUT algorithm [43], MAJORCLUST by Stein and Niggemann [91] and SPECTRAL by Shi and Malik [85].

In graph mining, association analysis is known mostly as frequent subgraph mining where the patterns that are searched for are subgraphs that occur often. Frequent subgraph mining was first introduced by Inokuchi et al. [40]. Other interesting algorithms are SUBDUE by Cook and Holder [16], FSG by Kuramochi and Karypis [53] and GSPAN by Yan and Han [101].

Graphs can have very specific shapes with their distinct properties. Data mining algorithms can aim to use these properties to their advantage. An example of such graphs are trees. In computer science, a tree is a data structure that is defined as a graph without cycles and where one vertex has been designated the root. The root defines a starting position which automatically gives the edges a natural direction towards or away from the root. Also, sometimes the branches have additional ordering. These properties make it easy to search in these trees and thus can the properties of trees be used by data mining algorithms to improve their performance or do things that are not possible in regular graphs. Since trees are used widely as data type, many data mining algorithms that work on trees have been designed. An overview of these algorithms can be found in the article by Nijssen et al. [73].

2.4 Hybrid Data

The previous sections described several different types of data. They can be divided into two main types of data: content-based data and contextual data. Content-based data tell something about a particular element in the data set, and contextual data tells something about how two different elements relate to each other. Content-based data are, for instance, single table data, or sets

of pictures, molecules, texts, or videos. Contextual data are, for instance, relational databases, graphs, or family trees.

Sometimes a specific dataset can be seen as content-based data, but also as contextual data. Consider, for instance, a database that consists of molecules. Whether this should be content-based or contextual data, depends on what the user sees as elements of interest. Suppose that this dataset consists of a list of molecules, and there is no data that gives any information about how one molecule relates to another. So, if the molecules are the elements of interest (e.g. because the user wants to classify individual molecules, or make groups of molecules that look alike) then the information in the data can be considered content-based. That is, each piece of data in the dataset only tells something about one specific molecule.

A molecule can be considered a type of graph. Here, the atoms can be considered the vertices and the chemical bonds connecting them can be considered the edges. Now, in this same dataset, the user wants to know something about atoms and bonds that appear in the dataset (for instance, which subgraphs of atoms appear often?). Now, the elements of interest are not the molecules, but the atoms. In this case, the bonds can be considered contextual information.

It is also possible that the dataset consists of a combination of the two types of data. In the case that a dataset has both content-based and contextual data, with regards to the same elements of interest, we define such a dataset as a *hybrid dataset*. Hybrid data is difficult to use in a data mining algorithm. An algorithm that is built for content-based data, cannot use the contextual information and vice versa. This means that a data mining algorithm that is created for a specific type of data cannot fully explore the data space of a hybrid dataset.

This problem can be overcome in different ways. One possibility is to create a completely new data mining algorithm that can use both content-based and contextual information. Another possibility is to adapt an already existing method that is created for one of the two types, so that it can use both types. The third option is to leave the core algorithm as it is, but alter the way that it handles the input data, so that it combines the two types of data and treats it as one. The method proposed in this thesis does exactly this. It inserts the contextual data into data mining algorithms that were designed for content-based data. In this way, no new algorithms need to be created, but in some cases, the used algorithm needs to be adjusted a bit. How this method works is described in more detail in Chapter 3.

2.5 Related Work

The idea to create a method such that hybrid data can be mined is not new, but not much work has been done in this direction yet. Probably the best known method that does so is Google's search engine. Although it is not usually

considered a data mining tool, it is still discussed here, showing that there actually is some sort of data mining component in it.

Two other methods incorporate content-based information in graph-based clustering tools. They are of course more related to the work in this thesis. The first is work of Neville et al. and the second is work of Zhou et al.

The Google Search Engine - PageRank

Although not usually considered a data mining tool, a closer look at the Google search engine will show there is an important data mining aspect to it. However, instead of it being one hybrid tool, it could better be considered as two different techniques that work independently.

Every time someone enters a query on `www.google.com`, first all the pages are filtered so that only those with some reference to the query remain, and then these remaining pages are ranked from most to least relevant for the query. Only pages that are in the Google data set can be presented as a result. It is not known precisely how big this data set is, but according to Lin et al. [59], the amount of pages exceeded 11.5 billion in February 2009. Starting with 25 million pages in 1998, Google managed a huge growth that nowadays exceeds this number, as the simplest queries (for instance, searching for all pages with 'a') result in finding more than 25 billion pages. Its popularity is especially shown in the amount of times it is used. Estimated by Sullivan to handle 2.8 billion queries per day in December 2009 [93] it had a worldwide share of 67% of all search queries.

The presented ranking is a combination of two rankings: one based on the structure of the internet, and one based on the content of the pages. They are created by independent techniques. Of these two techniques, the best known is the one that creates a ranking based on the structure. It is called PageRank, and considered as the core algorithm behind Google's search engine. Thoroughly described in the book by Langville and Meyer [55], in short, the algorithm works as follows. Every page x in the dataset gets a value, determining its rank. This value is called its PageRank, and denoted as $PR(x)$. It depends solely on its place in the structure defined by how pages link to each other. Every page can have inlinks, which are links that point from another page to it, and outlinks, which are links that point from the page to other pages. The PR of a page is the sum of the contributions of all inlinks to this page. This value is then divided over all its outlinks to determine the contribution of this page to all the pages it links to.

One can imagine that there are a lot of cycles in the graph structure formed by the internet. This will lead to many Catch-22 situations [35]. In order to calculate $PR(x)$ of a page x, one needs to know the PRs of all pages that link to it, but in order to calculate these, indirectly, $PR(x)$ needs to be known. The good thing is that in their paper, the creators of PageRank, Brin and Page, showed that it is possible to work around this problem [11].

First, the original adjacency matrix of the internet needs to be made stochastic and primitive. This can be done easily and without changing the nature of the adjacency matrix significantly. A more detailed description of the working of these adjustments and why they are needed, can be found in the book of Langville and Meyer [55]. When the matrix is made stochastic and primitive, starting with random values for all PRs and iteratively updating them, it will eventually result in a state where the values of the PRs do not significantly change anymore. This is a unique state, and this desired outcome will be reached after 50 to 60 iterations for the matrix that Google uses, despite what initial values were chosen.

In this way, Google can create a ranking of all indexed pages based on the structure of the web. Some alterations are made to reduce the ranking for sites that contain spam, but the general principle remains the same. Also for every page, Google keeps track of all words that appear in it along with a weight determining the importance of that word in that text. For instance, when a word appears frequently, or in the title, it gets a higher weight than a word that appears only once somewhere in the text. How this is done, is kept secret anxiously. Then, when someone enters a query in the Google search engine, all pages are ranked based on their PR and the weight of the words in the query.

So all pages are ranked based on the rankings of two independent techniques. One of these techniques, PageRank can be considered a data mining technique; it describes an underlying pattern of the structure by ranking the vertices in the graph in importance. Nonetheless, the techniques have no influence on each other, so calling the Google search engine one single hybrid technique is a bit far-fetched. It is therefore better to say that Google cleverishly combines the results of two different techniques: one data mining technique based on the structure of the internet and one technique based on the content of the pages.

The Method by Neville et al.

Neville et al. [72] were among the first to explicitly discuss the need for incorporating node content information into graph clustering. Their approach was to incorporate content-based information in existing graph clustering algorithms. More specifically, they consider graph clustering algorithms that can work with weighted edges, and define the weights according to the similarity of the nodes connected by the edge. Thus, they map the hybrid clustering problem to a graph clustering problem. After this mapping, any graph clustering method can be used.

In their paper they tested this approach for three different graph clustering algorithms. The first is Min-Cut by Karger [43]. This works by first placing all vertices in their own cluster and then repeatedly choosing an edge and merge the corresponding clusters, until only one edge remains. This is the minimal

cut that divides the graph in two clusters. This process can be repeated to use it as a divisive hierarchical clustering method.

Neville et al. adjusted this algorithm by assigning weights to the edges. These weights were defined by the similarity between the vertices connected by the edge. In the MIN-CUT-algorithm, the weights can be used as a probability for the edge that is to be chosen. In this case, edges between vertices that are very similar are more likely to be chosen than edges between vertices that are not alike. Thus, similar vertices have a greater chance of ending up in the same cluster.

The second algorithm used is MAJORCLUST by Stein and Niggemann [91]. This works by first assigning each vertex to its own cluster and then iteratively assigning each vertex to the cluster that is most prevalent among its neighbors. When more than one such cluster exists, a target cluster is chosen randomly between them. This is done until all vertices remain in the same cluster. Neville et al. altered this algorithm, by considering edge weights while selecting the most prevalent cluster.

The third algorithm that is extended is SPECTRAL by Shi and Malik [85]. Despite the fact that their field of research was image segmentation, they created a divisive hierarchical clustering algorithm for elements in a graph. The minimization of a self-defined, graph-based criterion was formulated as a generalized eigenvalue problem, after which the eigenvectors were used to create a partition of the data. Doing this recursively, creates a hierarchical clustering. Neville et al. used the similarities between elements to adjust the adjacency matrix used in the process.

In this way, Neville et al. created a method to include content-based data in a graph-based data mining tool. The downside of their method is that it can only include part of the content-based data. A similarity measure can calculate the similarity between any given pair of elements in the data set. Whereas all these similarities could be used in the data mining tool, the method of Neville et al. only uses the similarities between elements that share a relation, thereby ignoring similarities between unconnected elements. The method proposed in this thesis, on the other hand, merges all content-based and relational information. It is therefore to be expected that this could lead to better results.

The Method by Zhou et al.

More recently, Zhou et al. [102] proposed another method to include content-based information in already existing graph-clustering algorithms. In their method, they start with the original graph, as defined by the relational information in the data set. Every vertex in this graph is called a *structure vertex*. To this original graph, a new kind of vertices is included. They are called *attribute vertices*. For every different attribute value in the annotations of the structure vertices, one such attribute vertex is included. All structure vertices are connected to the attribute vertices that represent the attribute values that

are in the annotation of that structure vertex. An example of how this works can be seen in Figure 2.3.

In this way, the content-based information is thus inserted as structural information. The vertices in this newly obtained graph will then be clustered using a method based on k-medoids and a neighborhood random walk distance as distance measure. The exact working of k-medoids is described in Section 6.2.2. A *neighborhood random walk distance* between vertices v and w is defined as the chance that a random path with a given maximum length and starting in v will end in w.

This approach is somewhat more flexible with respect to trading off the different goals of standard clustering and graph clustering similarity; for instance, two nodes that originally did not have a path between them could still be in the same cluster, since they can be connected through one or more attribute nodes. This is not the case for most graph clustering methods.

A downside of this method is that there is a great chance that the ratio between the two kinds of information (content-based and relational) is very unbalanced. Consider, for instance, a data set with scientific papers where the citations are the relational information and the words that appear in the text are the content-based information. Normally, the amount of citations in a paper varies from a hand full to a few dozen, but the amount of different words in the text is in the range of several thousands.

This means that, when the method of Zhou et al. is applied here, every structure vertex will be connected to a lot of attribute vertices while the amount of other structure vertices it is connected to is hardly significant anymore. The effect of this will be that the amount of random paths that start with an attribute vertex will largely exceed the amount of random paths that start with a structure vertex. Accordingly, the influence of the content-based information will largely exceed the influence of the relational information. Thus, it can easily be that this method is unbalanced. For the method proposed in this thesis, on the other hand, the influences of these two different data types are much more balanced.

Chapter 3

Hybrid Similarity Measures

There is a need for generative methods to create hybrid data mining techniques. We propose a method that will allow the user to include relational information in any data mining tool that uses a content-based similarity or dissimilarity measure. We also show what the effect is of using this method on a distance or metric. The method is explained and formalized in two ways: one from a graph perspective and one from a matrix perspective.

3.1 Introduction

Data mining tries to find interesting patterns in large data sets. What patterns are interesting depends on the goals of the user. What these patterns look like depends on the data. Early data mining algorithms focus on content-based data. In a content-based dataset, all the data tells something about a single element and not how such an element relates to others. Despite the fact that it is not always written in that form, it could be seen as single table data: every row is an element, and every column is an attribute for those elements.

There are also datasets that do not fit in this form. Here, the information does not tell something about an element, but it tells something about how an element relates to its context. This is considered contextual data. Data mining algorithms designed for single table data are useless for contextual data. Therefore, this type of data requires the creation of a variety of data mining algorithms. Many come from the field of ILP or graph analysis.

There is also data that consists of both content-based and contextual data. A data mining algorithm that uses only one of the two types of information, cannot explore the complete data space. Therefore, there is a need for methods that work on such datasets and can explore the data space fully. Recently, Neville et al. [72] and Zhou et al. [102] created methods that insert content-based information in already existing contextual data mining techniques.

We propose a method that inserts contextual data in already existing data mining techniques that work on content-based data. It does so by adapting the similarity measure that the original data mining algorithm uses. In this chapter, the exact working of this hybrid similarity is explained in detail. The chapter is organised as follows. Section 3.2 explains the basic principle without going into technical details. Section 3.3 gives the formal definitions that describe the data for which this method is created. Section 3.4 defines the proposed hybrid similarities from a graph-based point of view, and describes how they relate to the original, content-based similarity. Section 3.5 describes the same hybrid similarities, but now from a matrix-based point of view. Section 3.6 explains how these hybrid similarities are plugged into the existing data mining algorithm. Finally, Section 3.7 gives some advice on the situation in which it is useful to use any of the hybrid similarities.

3.2 The Basic Principle

A data set comprises of a set of "objects of interest," which are the objects about which the data set is (e.g. scientific papers, people, purchases, et cetera). These objects will be referred to as "elements." Besides these elements, a data set normally contains information that tells us more about these elements. In this case, two different types of information are distinguished: information that tells the user more about a certain element, and information that tells the user more about how a certain element relates to other elements. The first type will be known as the *content-based* information and the second type will be known as the *contextual* information.

To clarify this a bit more, consider a database of scientific papers with their texts and citations. The objects of interest, or elements, are the scientific papers. A data mining tool therefore should be saying something about a paper, or a group of papers. Examples can be: "this paper is about that subject," or "these papers are very similar and thus probably about the same subject." The content-based information in this case is the text of the paper and the relational information is the information about the citations.

As stated in Chapter 2, it is not so straightforward to combine these two types of information, and therefore data mining tools typically use only one of the two. The method used in this thesis combines the two types of information indirectly. To do so, it assumes there is a similarity or dissimilarity measure that can tell how much any two elements look like each other. With a similarity measure, a high value between two elements means they are very similar and thus look a lot like each other. A dissimilarity measure works in the exact opposite way: the lower its value, the more two elements are alike. Despite this difference in meaning between high and low values for similarity and dissimilarity measures, they act in a similar way. So from now on, we consider similarity measures only, but the same story could be told for dissimilarity measures.

To summarize, there is a data set with elements that have content-based information that can be quantified by a similarity measure expressing how similar two elements are, and relational information between these elements. Now, a hybrid similarity measure can be constructed by regarding the similarities from one element to the "neighborhood" of another. This neighborhood is defined by the contextual information in the dataset. The similarity between one element and the neighborhood of the other is defined by the content-based information in the dataset. Therefore, it is clear that the hybrid similarity measure uses both types of information.

This is visualised in Figure 3.1 where the hybrid similarity between two elements v and w can be calculated. Two types of information are drawn in this figure. The contextual information is drawn by a solid line indicating that there is a relation between these two elements. Elements that are related are known also as "neighbors." The neighborhood of an element is the set of all its neighbors. Thus, in this case, the neighborhood of v is $\{a, b\}$ and the neighborhood of w is $\{c, d, e, f\}$. The content-based information is illustrated in this figure by either the similarity between v and w (dashed line) or the similarity between v and the neighbors of w (dotted lines).

Indirectly, the two types of information are used. The relational information is used to determine which are the neighbors of an element. The content-based information then is used to calculate the similarity between an element and the neighbors of the other element.

3.3 The Data Format: Annotated Graphs

Consider the dataset that needs to be clustered to be an annotated graph. This data set D can be defined as $D = (V, E, \alpha, \lambda)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices or elements, $E \subseteq V \times V$ is the set of edges, $\alpha : V \to \mathcal{A}$ a function that assigns to any vertex v of V an "annotation", and $\lambda : V \to \mathcal{L}$ a function that assigns to any vertex v of V a "label". The annotation $\alpha(v)$ is considered to be the *content* of vertex v and the label $\lambda(v)$ is considered to be the class of vertex v.

The graph is undirected and an edge cannot loop back to the same vertex, so with two vertices $v, w \in V$ this means that $(v, w) \in E \Leftrightarrow (w, v) \in E$ and $(v, v) \notin E$. Furthermore, for every vertex $v \in V$, the neighborhood of that vertex is defined as $\mathcal{N}(v)$, where $\mathcal{N}(v)$ is the set of vertices that are connected to v with an edge. So with two vertices $v, w \in V$, $w \in \mathcal{N}(v) \Leftrightarrow (v, w) \in E$. The amount of neighbors of v can be seen as the size of $\mathcal{N}(v)$ which is denoted as $|\mathcal{N}(v)|$, and since there are no loops ($(v, v) \notin E$), the amount of neighbors of v is the same as the degree of v, denoted as $\text{DEG}(v)$. Furthermore, every element has at least one neighbor, so $\mathcal{N}(v) \neq \emptyset$, and thus $|\mathcal{N}(v)| > 0$

The space of possible annotations is left open; it can be a set of symbols from, or strings over, a finite alphabet; the set of reals; an n-dimensional Eu-

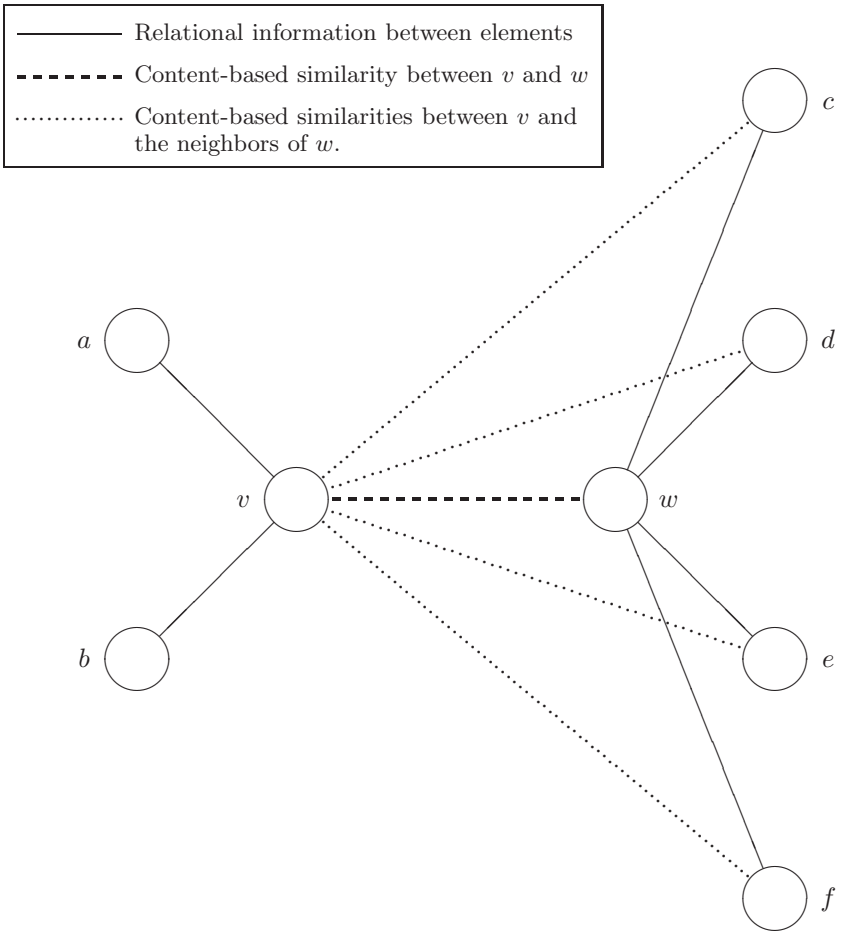


Figure 3.1: Visualisation of the basic principle behind the hybrid similarity. In this picture, the solid lines are the relations as described by the contextual information in the database. Thus, a and b are the neighbors of v, and c, d, e, and f are the neighbors of w. When calculating the similarity between v and w, a data mining tool designed for data in the single table form would only consider the content-based similarity between v and w (dashed line). A graph mining tool, on the other hand, only would consider the relational information (solid lines). The hybrid method regards the content-based similarities from one node to the neighbors of the other (dotted lines). In this picture, for clarity reasons, the content-based similarities between w and the neighbors of v (a and b) are left out.

clidean space; a powerset of one of the sets just mentioned; etc. The only constraint on $\mathcal{A}$ is that it must be possible to define a similarity measure $\mathcal{S} : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$ as a function that assigns a value to any pair of annotations expressing the similarity between these annotations. Since this similarity is entirely based on the content of the vertices, it will be called the *content-based similarity*, or $\mathcal{S}_{content}$. Normally the value of this similarity is in the range $[0, 1]$ where 0 stands for no similarity at all and 1 means that the elements are considered to be identical.

The space of possible labels is defined more strictly. It is a finite, discrete set, so $\mathcal{L} = \{L_1, L_2, \ldots, L_l\}$. The label of a vertex can be seen as its class. With supervised learning, this label is given and it is the task of the DM tool to learn a relation between V, E and α on one side and λ on the other. With semi-supervised learning, part of λ is given and the rest is unknown or hidden. The DM tool then needs to predict this rest, given V, E, α and part of λ. With unsupervised learning, λ is completely unknown or hidden and the DM tool needs to predict it, given V, E and α. When information about λ is hidden, it can be used to check the quality of the found solution after the DM tool has finished.

3.4 Similarity Measures

This section defines the main similarity measures as used in this thesis.

3.4.1 Content-based Similarity

As said, we assume a similarity measure $\mathcal{S} : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}$ is defined over the space of annotations $\mathcal{A}$. From this, we can immediately derive a first similarity measure on the nodes of the graph, which we call the content-based similarity:

$$\mathcal{S}_{content} : V \times V \rightarrow \mathbb{R} : \mathcal{S}_{content}(v, w) = \mathcal{S}(\alpha(v), \alpha(w)) \tag{3.1}$$

In other words, the content-based similarity, $\mathcal{S}_{content}$, is a similarity that is based purely on the content, or annotations, of the vertices. In general, this is the similarity measure that originally is used in data mining tools as described in Section 2.2. In Figure 3.1, it is illustrated by the dashed line between v and w. There, it is also clear, that no relational information is used to compute this similarity.

3.4.2 Contextual Similarity

To define the contextual similarity, it is necessary to first define the *neighborhood similarity* $\mathcal{S}_{neighbor} : V \times V \rightarrow \mathbb{R}$ between two vertices v and w from V,

as the average content-based similarity between v and all neighbors of w:

$$\mathcal{S}_{neighbor}(v,w) = \frac{\sum_{u \in \mathcal{N}(w)} \mathcal{S}_{content}(v,u)}{|\mathcal{N}(w)|} \tag{3.2}$$

In Figure 3.1 this similarity is computed as the average of the dotted similarities. It is not symmetric, but it can easily be symmetrized. This leads to the *contextual similarity* $\mathcal{S}_{context} : V \times V \to \mathbb{R}$:

$$\mathcal{S}_{context}(v,w) = \frac{\mathcal{S}_{neighbor}(v,w) + \mathcal{S}_{neighbor}(w,v)}{2} \tag{3.3}$$

The motivation behind this similarity measure is that, if similar nodes tend to be linked together, then the neighbors of w in general are similar to w. Hence, if similarity is transitive, a high similarity between v and many neighbors of w increases the reasons to believe that v is similar to w, even if there is little evidence of such similarity when comparing v and w directly (for instance, due to noise or missing information in the annotation of w).

3.4.3 Combined Similarity

The contextual similarity measure from (3.3) is complementary to the content-based similarity from (3.1); it does not use the content-based similarity between v and w at all. This can be seen easily in Figure 3.1. The content-based similarity is represented by the dashed line and the contextual similarity by the dotted lines, of which it is the average. The dotted lines are derived from the relational information drawn by the solid lines, but the dashed line is never used. In practical settings, it may be good not to ignore the content-based similarity entirely, so Witsenburg and Blockeel [99] also introduced the weighted average of the content-based similarity and the contextual similarity as the *combined similarity* $\mathcal{S}_{combined} : V \times V \to \mathbb{R}$:

$$\mathcal{S}_{combined}(v,w) = c \cdot \mathcal{S}_{content}(v,w) + (1-c) \cdot \mathcal{S}_{context}(v,w) \tag{3.4}$$

with $0 \leq c \leq 1$. The constant c determines the weight of the content-based similarity in the combined similarity. As Witsenburg and Blockeel [99] found no strong effect of using different values for c, from now on we consider only the combined similarity with $c = \frac{1}{2}$.

Both the contextual and the combined similarity measures are hybrid similarity measures, since they use both content-based and graph information. Whereas the contextual similarity measure between two nodes v and w does take the contents of v and w into account, it just does not use the direct similarity between these contents.

Note that any standard clustering method that can cluster nodes based on (only) their content similarity can also cluster nodes based on the contextual or

combined similarity, and in the latter case it implicitly takes the graph structure into account; the method itself does not need to be altered to be able to process graph data.

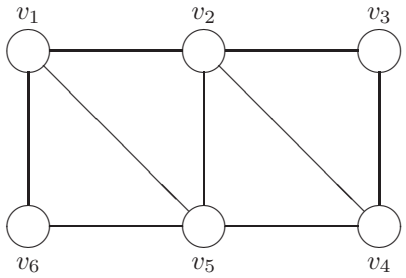
3.5 Matrix-based Computations

In this section, the contextual similarity will be explained in the light of matrix multiplication, to create a broader perspective on the hybrid similarities.

Considering the dataset $D = (V, E, \alpha, \lambda)$ as described in Section 3.3, it can be seen that there are two types of information in it: content-based information and relational information. The content-based information is defined by the function α that assigns an annotation to each element. The relational information is defined by the set of edges E, which defines whether there is a relation between two elements or not.

From these two types of information, two different $n \times n$ matrices can be constructed (where n is the amount of vertices in V, or the amount of elements in the dataset D). The first matrix would be the similarity matrix S, where every element s_{ij} is the content-based similarity between v_i and v_j: $s_{ij} = \mathcal{S}_{content}(v_i, v_j)$. The second matrix is the adjacency matrix A where each element a_{ij} is defined by:

$$a_{ij} = \begin{cases} 0 & \text{if } (v_i, v_j) \notin E \\ 1 & \text{if } (v_i, v_j) \in E \end{cases} \tag{3.5}$$



	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	1	0	0	1	1
v_2	1	0	1	1	1	0
v_3	0	1	0	1	0	0
v_4	0	1	1	0	1	0
v_5	1	1	0	1	0	1
v_6	1	0	0	0	1	0

Figure 3.2: Example of a graph and its adjacency matrix

Figure 3.2 shows an example of an undirected and unweighted graph, and its adjacency matrix. The adjacency matrix has one feature that is aspecially interesting for calculating the hybrid similarity measure. When the adjacency matrix is multiplied with itself, $A^2 = A \times A$, then every element a'_{ij} in A^2 with $i \neq j$ gives the amount of paths from v_i to v_j with length 2. An element a'_{ii} in A^2 gives the amount of neighbors of v_i.

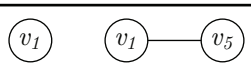
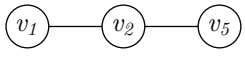
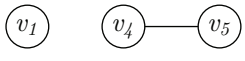
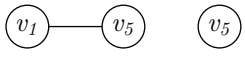
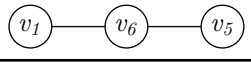
k	a_{ik}	a_{kj}	$a_{ik} \cdot a_{ik}$	*graph representation*
1	0	1	0	
2	1	1	1	
3	0	0	0	
4	0	1	0	
5	1	0	0	
6	1	1	1	

Table 3.1: Visualisation of how an element a'_{ij} from A^2 denotes the amount of paths of length 2 in the graph represented by A. In this example $i = 1$ and $j = 5$. The first column shows k which in the sum goes from 1 to $n = 6$. The second and third column show the corresponding a_{ik} and a_{kj} respectively. The fourth column shows the product of a_{ik} and a_{kj}. The fifth column shows the edges from the graph that match the a_{ik} and a_{kj} from the second and third column. It clearly shows that the fourth column gives all the terms of the sum from (3.6) and that each of those terms is one whenever there is a path through the corresponding v_k.

To illustrate this, consider that each element a'_{ij} in A^2 is calculated with

$$a'_{ij} = \sum_{k=1}^{n} a_{ik} \cdot a_{kj} \tag{3.6}$$

In the case of the graph from Figure 3.2, we have $n = 6$. In the same figure it can be seen that an element in A is either 0 (no edge) or 1 (edge), so each term in the sum from (3.6) is 1 when both a_{ik} and a_{kj} are 1, and 0 otherwise. In the graph, this means that such term is 1 for every k where the edges (v_i, v_k) and (v_k, v_j) both exist, and thus, the sum counts the amount of vertices that are connected to both v_i and v_j. There exists a path of length 2 from v_i to v_j if, and only if, there is a vertex that is connected to both v_i and v_j. So counting the number of vertices that are connected to both v_i and v_j is the same as counting the number of paths of length 2 from v_i to v_j, and thus every element a'_{ij} in A^2 gives the amount of paths of length 2 between v_i and v_j. This principle is also illustrated in Table 3.1.

Regarding how (3.6) was converted to the drawing in the fifth column of Table 3.1, it is easy to see how on the main diagonal of A^2 the degrees of the vertices arise. First, take a closer look at this drawing. Figure 3.3 shows it again,

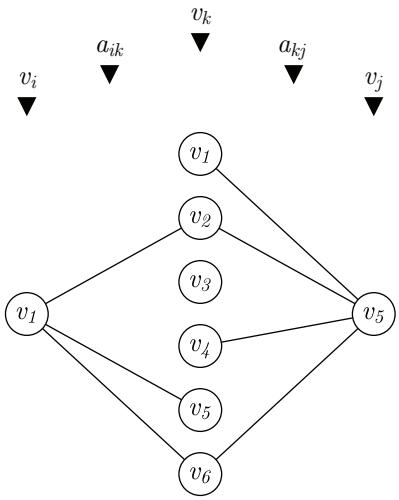


Figure 3.3: This drawing is created from the fifth column of Table 3.1 by merging all begin and end vertices (v_1 and v_5 respectively). Also the corresponding labels for the calculation of a_{15} from A^2 are added. On the far left and far right, v_i and v_j respectively, which in this case are v_1 and v_5. In the middle, the column of vertices comes from k going from 1 to n. Vertices v_i and v_j are connected to the vertices in the center column when they are connected to them in the graph.

but this time with labels indicating what part of the matrix multiplication is represented by which element in the drawing. Thus, it is easy to see what the effect is of substituting an element in (3.6).

For instance, the effect of substituting v_j with v_i and adjusting the connecting edges accordingly, is shown in Figure 3.4. This is the same effect as substituting v_j with v_i in (3.6). So calculating an element a'_{ii} in A^2 results in Figure 3.4 (right). Here it is easy to see that on the main diagonal in A^2 the degrees of the vertices arise. The result is that the amount of paths of length 2 in Figure 3.4 (right) is therefore the same as the amount of vertices connected to v_i.

It is also possible to substitute one of the matrices from the multiplication. In that case the edges connecting either one or the other vertex come from this new matrix. To ensure that the result is useful, this new matrix needs to have the same "shape" as A (i.e. every row or column must refer to the vertex in the graph as that row or column would do in A). The similarity matrix S is therefore a suitable candidate. Both the first and the last 'A' in '$A \times A$' can be substituted, resulting in either '$A \times S$', or '$S \times A$'. Figure 3.5 shows the effect of these substitutions. A comparison between these pictures and Figure 3.1 shows the resemblance between the two.

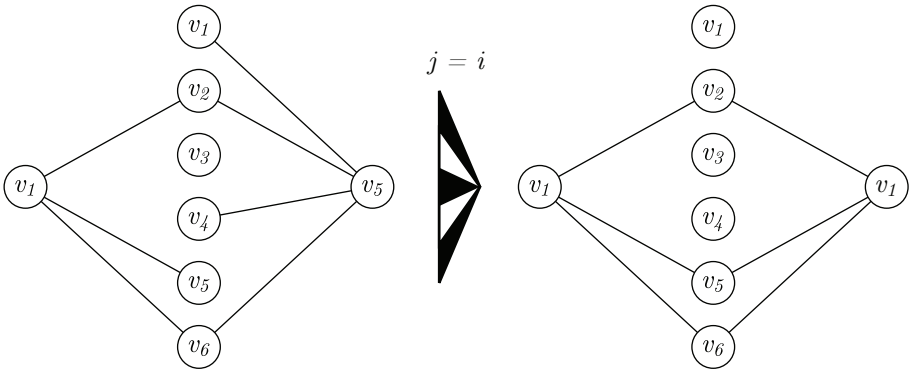


Figure 3.4: Taking Figure 3.3 (on the left), substituting v_j with v_i, and adjusting the edges connecting the second v_i accordingly, results in the picture on the right. Here, it is easy to see how on the main diagonal of A^2 the degrees of the vertices arise. This is simply counting the paths of length 2 that have occurred, which is the same as the amount of vertices connected to v_i.

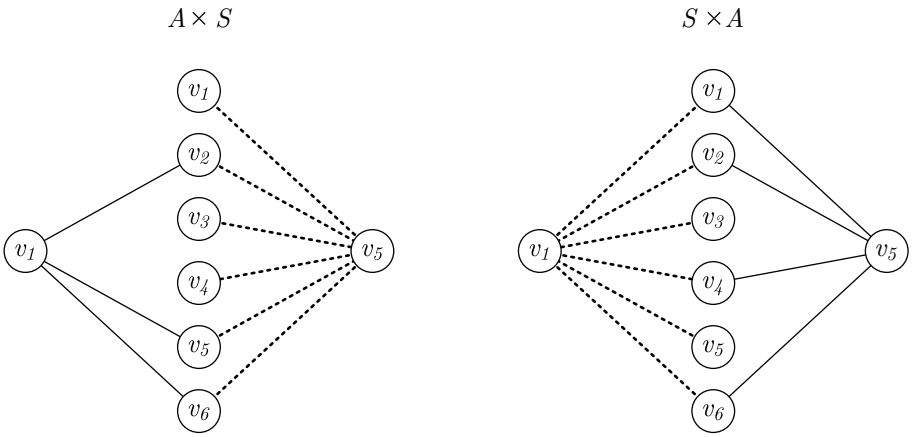


Figure 3.5: The result of substituting a_{kj} with s_{kj} (left) and a_{ik} with s_{ik} (right). Both substitutions come from Figure 3.3.

As the edges have been replaced, again, the amount of paths of length 2 needs to be counted. But now, any such path is a combination of an edge from the graph and a similarity from S. To be more precise, any element x_{ij} from $A \times S$ is the sum of the similarities between v_j and all neighbors of v_i, and any element x_{ij} from $S \times A$ is the sum of the similarities between v_i and all neighbors of v_j. This is very close to the contextual similarity from (3.3), but that is the average of all similarities instead of the sum.

So, to formulate the similarities from Section 3.4 with matrices, only a little adjustement from combining $A \times S$ with $S \times A$ is needed. In order to come to the average similarity instead of the sum, one needs to divide all values in the resulting matrix by the degree of the vertex whose neighborhood is used. This is the vertex that corresponds to the row or column in the adjacency matrix that is used for calculating the element in the resulting matrix. Thus, every element in the i-th row of the result of $A \times S$ needs to be divided by the degree of v_i, and every element in the j-th column of the result of $S \times A$ needs to be divided by the degree of v_j.

Dividing by a number is the same as multiplying with its inverse. Therefore, consider a neighbor matrix N, which is a diagonal matrix where each element n_{ii} is the *neighbor factor* and is one divided by the number of neighbors of v_i. Now, the contextual similarity from the previous section can also be computed by:

$$\frac{N \times (A \times S) + (S \times A) \times N}{2} \tag{3.7}$$

The hybrid similarity measure is therefore a method that multiplies the adjacency matrix with the similarity matrix and vice versa, and it uses these as similarity measures for the data mining tool. The only thing that needs to be done is a small adjustment in the form of taking average values (to make sure all values stay in the same range).

3.6 Plug and Play

The method proposed in this thesis is not a new data mining tool. Instead of that, it can be seen as a method to add relational information to an already existing data mining tool that only uses content-based information. To illustrate this idea, first consider Figure 3.6. This figure shows a data mining tool that can be considered as a black box; it is not important to know how it works precisely. The only features that are important are that it only uses the content-based information from the data set as input and some similarity measure that is based only upon that. With that, the data mining tool comes to its results.

Many data mining tools work regardless of which exact distance or similarity measure is used. For instance, when the content-based information in the data set consists of vectors, a data mining tool that uses the cosine between two vectors as the similarity, also could use the Jaccard index or Manhattan distance

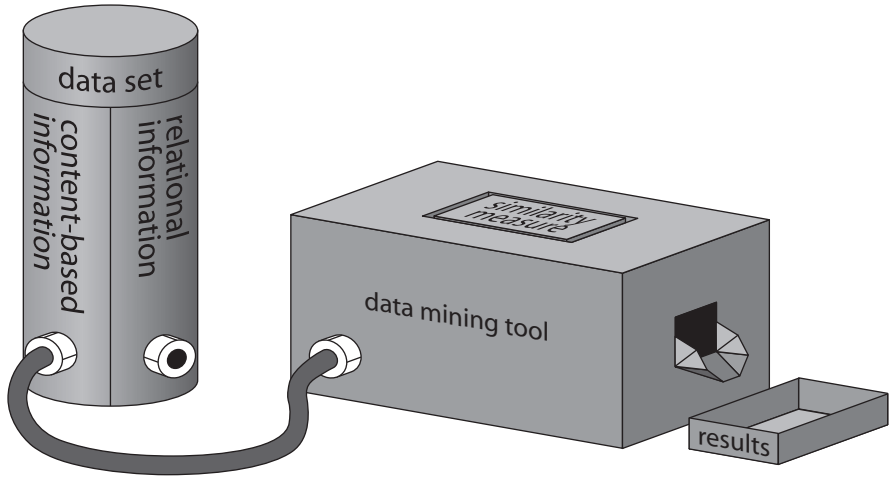


Figure 3.6: A data mining tool can be considered as a black box. The content-based information from the data set is used as an input leading to results as an output. The data mining tool uses some similarity measure based only on the content-based information from the data set. The relational information from the data set is not used.

between them. This principle is illustrated in Figure 3.7. The similarity measure can be seen as a plugin for the data mining tool. For the data mining tool to work, one needs to plugin a similarity measure. This can be anything as long as it fits the data. This similarity measure can therefore be regarded as independent of the data mining tool.

Instead of plugging the preferred content-based similarity directly into the data mining tool, one can also choose to plug this content-based similarity into the hybrid similarity measure from this thesis. At this point, one can use this result as similarity measure for the data mining tool. This principle is illustrated in Figure 3.8. Any similarity measure that can be plugged into the data mining tool can be plugged also into the hybrid similarity measure. This combines the content-based similarity measure with the relational information from the data set to come to a new similarity measure. Then, this new similarity measure can be used in the data mining tool in the same way that the content-based similarity would have been used. The data mining tool does not need to be adjusted for this.

One benefit of the hybrid similarity measure is the fact that the user does not need any extra domain specific knowledge to use it. This cannot always be said of content-based measures. For instance, Monge and Elkan use an alphanumeric edit distance to identify duplicate alphanumeric records [68]. Also,

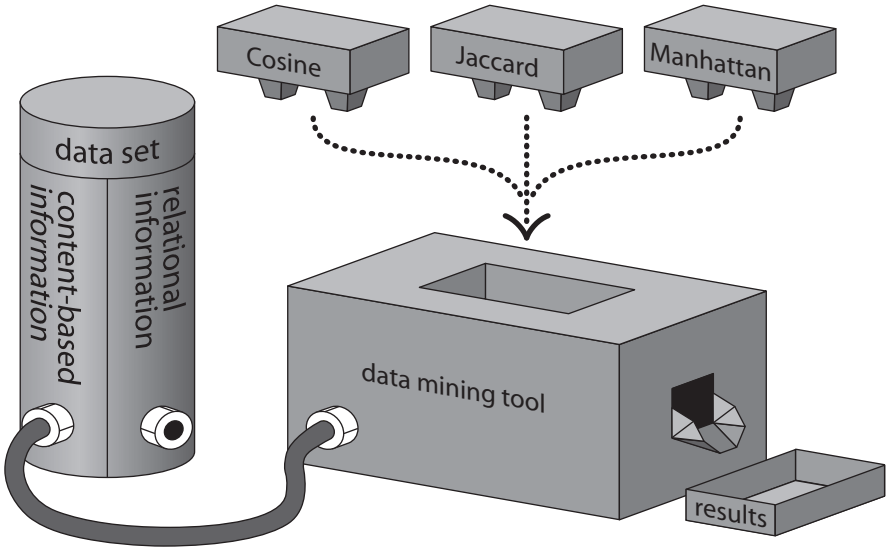


Figure 3.7: A data mining tool can use different similarity measures. For instance, a data mining tool that uses the cosine similarity, would work according to the same principles when using the Manhattan distance or the Jaccard index.

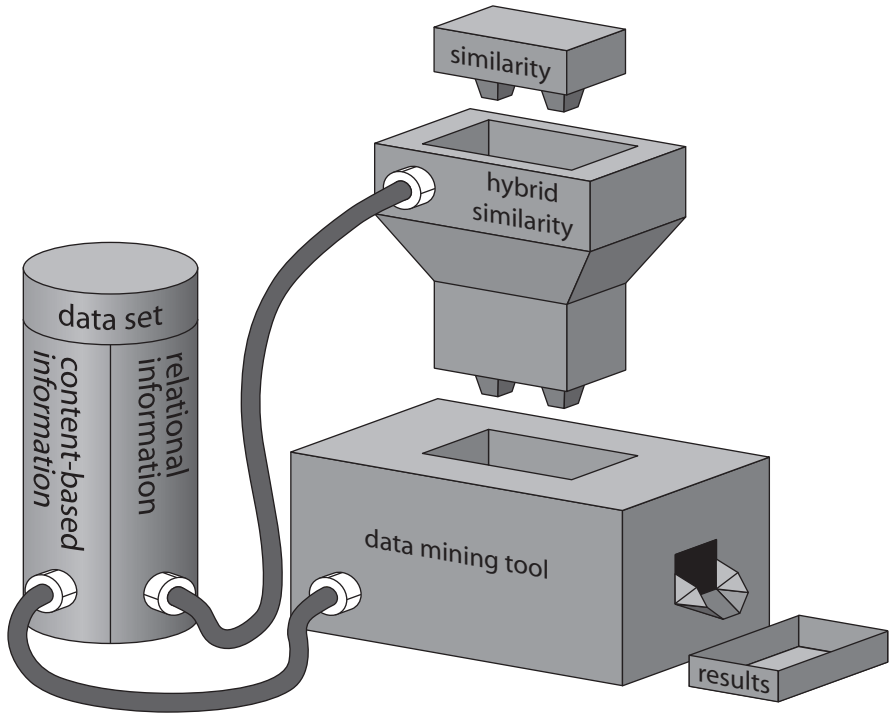


Figure 3.8: The hybrid similarity measure proposed in this thesis can be seen as a plugin for a data mining tool. Instead of plugging a content-based similarity measure directly into the data mining tool, this measure can be plugged into the hybrid similarity measure. The hybrid similarity measure combines the content-based similarity measure with the relational information from the data set. In this way, it comes to a new similarity measure that can be used by the data mining tool in a similar way as it would have used the content-based similarity.

data mining on protein structures is often done using a distance function that rotates and translates structures to superimpose them. If the user understands the content-based similarity, then this is enough to use the hybrid similarity.

3.7 Using the Right Content-based Similarity

The previous section showed that the hybrid similarity measure can replace the content-based similarity. This can always be done. However, to increase the chance that the extra effort and computing time will indeed lead to better results, one needs to make sure that the original similarity measure is not based upon the relational information. As an illustration, consider a data mining tool that uses a distance measure that takes the length of the shortest path between two elements in an unweighted and undirected graph as the distance between them. In this case, using the hybrid similarity does not add much to the original similarity. This can easily be seen.

Assume the distance between two elements v and w, as defined by the length of the path between them, is p. To calculate the contextual distance between v and w, the distances between v and all neighbors of w and between w and all neighbors of v need to be aggregated. These distances are limited in comparison to the distance p. Since a neighbor of v or w is, by definition, respectively connected to v or w, the maximum length for any of these distances is $p+1$. In a similar way, it can be concluded that the minimum length of these distances is $p-1$. Thus, all distances between v and the neighbors of w, or between w and the neighbors of v are in the range $\{p-1, p, p+1\}$.

This means that the contextual distance between v and w is highly dependent on the original distance between v and w. Therefore, one can safely conclude that it is to be expected that the results with the contextual distance will not differ a lot from using only the original distance. Hence, the added value of the hybrid similarity is extremely small. So, if we want the hybrid similarity to actually add something to the original distance, we should take care that the original distance is not based on relational information.

Chapter 4

Motivation

We give motivation for the potential improvement of the hybrid similarities in comparison to a similarity measure that is only based on the content. For many different reasons, it is difficult for a similarity measure to get the semantic similarity between elements precisely. When calculating the similarity between elements, the hybrid similarities explore a greater part of the data space. This creates a more confident value for a similarity between two elements.

4.1 Introduction

In general, data mining algorithms are designed for a specific type of data: content-based data or contextual data. Only recently, methods are proposed that can work on hybrid data (i.e. data that consists of both types). In this thesis we propose a new method for this task. This method inserts contextual information in a data mining algorithm that is designed for content-based information.

As proposed in this thesis, the contextual similarity between two elements is the average, content-based similarity of these elements and the neighbors of the other. This thesis also proposes that the combined similarity is the average of the contextual similarity and the original, content-based similarity between the two elements. From a matrix perspective, these new similarities are created by multiplying the adjacency matrix of the graph with the similarity matrix of the elements, and vice versa, and then adjusting the values for the size of the neighborhood.

This chapter gives a theoretical motivation for why we expect the hybrid similarities could outperform the content-based similarity. We also try to formulate characteristics for datasets where it could be beneficial to use the hybrid similarities. The rest of this chapter is organized as follows: Section 4.2 describes three characteristics of datasets. Section 4.3 describes the principles

about distance and similarity measures. Section 4.4 describes the difference between semantic and measured similarities. Section 4.5 combines all these characteristics and principles in an explanation on why the hybrid similarity will, most likely, outperform the original similarity. Finally, Section 4.6 gives a summary of this chapter.

4.2 Important Characteristics for Networks

To understand the working of the hybrid similarity measure, it is necessary to first take a closer look at three possible characteristics of network based databases that play a big role in this. These are homophily, transitivity of similarity, and content variability.

Homophily

In a network or linked dataset, *homophily* is the characteristic that can be described as the tendency of elements that are similar to be connected. One example is a dataset that consists of scientific papers. In this dataset is, amongst others, information about which paper cites which other paper. The latter can be seen as relational information, for it links two papers when one cites the other. In general, most citations in a scientific paper are to papers that are (roughly) about the same subject. Thus, a set of papers about the same subject will be highly connected.

Another example is a social network. Here, the relations are defined by the ability of users to indicate which other users are their friends. Often, friends of friends have a good chance of becoming your friends as well, thus creating highly connected groups of friends. Also, friends tend to like the same things (e.g. movies, books, activities). This goes in two directions: friends of someone share their interests with him and then he gets excited by it himself, or someone goes to an event for people with a certain interest and meets people there who become his friends.

Overall, there are many situations where similar items tend to be highly connected. Also, the neighborhood of an element tends to consist of elements that are fairly similar. Of course, it can always happen that highly dissimilar elements are connected or that highly similar elements are not connected.

Transitivity of Similarity

Given a domain and a similarity measure, *transitivity of similarity* can be described as the fact that when two elements are similar, a third element that is similar to one of the two, is also fairly similar to the other. If the chosen similarity measure does not follow this rule, very undesirable things can happen. For instance, consider three elements, u, v and w, where v is very similar to

both u and w, but u and w are very dissimilar to each other. A clustering algorithm could place u and w in the same cluster as v since they are very similar. However, this means that u and w are in the same cluster, which is highly unwanted since they are very dissimilar.

When a distance measure is used instead of a similarity measure, the principle is shown in the triangular inequality. It states that for any three elements u, v and w, it should hold that $d(u, w) \leq d(u, v) + d(v, w)$, where $d(v, w)$ denotes the distance between v and w.

Let the reader be advised that this is not exactly the same as 'transitivity of equality', which is precisely formulated as '$a = b \wedge b = c \Rightarrow a = c$.' Transitivity of similarity, on the other hand, can be formalized as '$a \sim b \wedge b \sim c \Rightarrow a \approx c$'.

Content Variability

The contents of nodes from the same class may differ, and in most real life data where elements are characterized by their content, they do so. Consider, for instance, a database with texts about various subjects. Amongst it might be some texts about football players. All of these texts about football players will be different, but some words that are related to football (e.g. cup, goal, and league) will, on average, appear more often in those texts than in other texts. To illustrate this, Table 4.1 shows how often certain words that are highly related to football appear in five texts about football players.

The amount of times a particular word about football appears in such a text is obviously not the same for all texts. However, regarding all texts about football players, there is some sort of average amount of times that certain words appear in them.

It is obvious that even for words that are related to football, the amount of times they appear in a text about a football player will differ. However, in general, those words appear more often. From this, as one takes all texts about football players, it is possible to create a mean vector that gives for each word the most typical amount of times it appears in a text about a football player. The actual amount of times it appears in a specific text about a football player can of course highly differ from this average. But, in general, the keyword vector of a text about a football player will more or less point in the same direction as this mean keyword vector. *Content variability* refers to how much the content of an individual element may differ from the mean content of all elements in the same class.

4.3 Distances, Similarities, and Dissimilarities

The hybrid similarity works for data mining tools that use some kind of measurement to compare two elements with each other. This measurement should state how much these element look alike. This can of course be done in many

Keyword	Mean	Best	Daei	Di Stéfano	Kuijt	Sócrates
ball	1.8	8	0	0	1	0
career	9.4	11	12	8	9	7
club	19.0	16	22	21	28	8
coach	4.4	1	17	2	1	1
competition	3.6	3	7	2	6	0
cup	39.4	16	98	37	31	15
draw	1.2	1	1	0	3	1
stadium	7.8	2	0	24	13	0
fans	2.0	1	5	1	2	1
fifa	10.4	5	32	10	2	3
final	5.8	3	4	6	15	1
football	11.4	16	22	7	3	9
footballer	5.4	9	3	6	2	7
game	11.2	8	13	3	28	4
goal	34.4	25	45	9	84	9
international	12.4	3	29	20	9	1
league	21.0	14	35	10	32	14
manager	5.8	7	13	6	3	0
match	16.2	18	32	11	18	2
opponent	1.2	2	1	1	2	0
pass	1.2	1	1	0	2	2
penalty	3.0	1	0	0	14	0
pitch	2.2	4	2	1	1	3
play	17.0	22	16	17	19	11
player	10.6	11	16	14	7	5
professional	2.0	3	1	0	3	3
qualification	17.4	9	47	7	23	1
result	2.4	4	4	1	3	0
scorer	4.2	2	9	4	5	1
score	20.4	18	20	4	58	2
season	14.2	15	12	5	38	1
squad	2.4	2	3	0	6	1
striker	2.6	2	1	0	9	1
supporter	1.0	0	1	0	1	3
team	14.2	9	34	8	15	5
tournament	1.6	0	4	1	3	0
uefa	3.2	5	5	3	3	0
victory	2.2	1	0	3	7	0
win	14.2	11	9	14	34	3

Table 4.1: Example of the amount of keywords in different texts about football players. This table is created using the wikipedia pages of five football players: George Best, Ali Daei, Alfredo Di Stéfano, Dirk Kuijt and Sócrates.

different ways, but three categories can be distinguished: distances, similarities, and dissimilarities.

A *distance measure* calculates a numeric value that indicates how close or how far two elements are from each other. The smaller the distance between two elements, the closer they are, and thus the more they are considered to look alike. Distance measures in general have no maximum. This means there is no strict upper boundary for a distance measure. In practice, however, the distance is most of the time limited by the domain of the data set.

More formally, a distance measure is defined as a metric. A *metric* is defined as a function $d : V \times V \to \mathbb{R}$ that for all $u, v, w \in V$ meets four properties:

1. *Non-negativity* property: all distances are positive, so $d(v, w) \geq 0$.

2. *Identity* property: when two elements have a distance of 0 then, and only then, are they considered the same. Therefore, $d(v, w) = 0 \Leftrightarrow v = w$.

3. *Symmetry* property: the distance from one element to another is the same as the distance in the opposite direction, so $d(v, w) = d(w, v)$.

4. *Triangle inequality* property: when going from v to w directly, the distance travelled is always shorter than, or equal to the distance travelled when going from v to w through u, so $d(v, w) \leq d(v, u) + d(u, w)$.

The triangle inequality is an important property for a distance measure. This principle has been used by Elkan to seriously speed-up k-means [21], and by Kryszkiewicz and Lasek to improve the efficiency of DBSCAN [52]. Both use the fact that when $d(v, u)$ and $d(u, w)$ are known, an upper boundary for $d(v, w)$ is defined by $d(v, u) + d(u, w)$ without need to calculate $d(v, w)$. Also, Baraty et al. [3] showed that improper use of data mining tools in an environment where there is no triangular inequality could result in highly unwanted outcomes.

Another way to approach the comparison of two elements is not by looking at the distance between them, but rather by comparing how similar they are. This can be done with a *similarity measure*, which is a function $\mathcal{S} : V \times V \to \mathbb{R}$ that for all pairs of elements in V states how similar they are. A high value for a similarity means that they are very similar, which would correspond to a low value for a distance and vice versa. Nevertheless, it is not possible to state that a similarity is the exact opposite of a distance. The four conditions previously described for a metric do not necessarily hold for the inverse of a similarity measure.

The only condition that could be fulfilled by a similarity measure is that its value is always greater than or equal to 0. Any similarity measure used in this thesis is also symmetric, but this is not a constraint in general. The benefit of these constraints is that clever use can speed up many data mining algorithms. The benefit of not fulfilling these constraints is that this leaves a much wider range of possible similarity measures.

	decreasing value means elements are more alike	*increasing value means elements are more alike*
does not meet the properties of a metric	dissimilarity	similarity
meets the properties of a metric	distance	

Table 4.2: Overview of the important differences between distances, similarities and dissimilarities.

Thus, a similarity measure differs from a distance measure in two important ways. First, it does not meet the properties of a metric. Second, an increasing value for a similarity measure means that the elements are more similar, while for a distance measure a decreasing value means that the elements are closer. Sometimes an algorithm needs that an increasing value of the measurement means that the elements are more distant, but the domain of the dataset does not allow to define a metric that meets all four properties. When it is not necessary that the algorithm has these properties, then the user can choose to use a dissimilarity measure (which is basically the opposite of a similarity measure). A *dissimilarity measure* is a function $\mathcal{D} : V \times V \rightarrow \mathbb{R}$ where a low value between one pair of elements means that they are closer than a pair of elements with a high value for $\mathcal{D}$. It does not necessarily meet the properties that would make it a metric.

To summarize, a distance measure differs from both a similarity measure and a dissimilarity measure in that it meets the four properties of a metric. A similarity measure differs from both a distance measure and a dissimilarity measure in that an increasing value means that the two elements are more alike. The differences between the three concepts are schematically described in Table 4.2.

It is easy to see that it is not possible to create a measurement where an increasing value means that elements are more alike, and that meets the properties for a metric. According to these properties, all values must be greater than or equal to 0, and if, and only if, an element is compared to itself, this value must be 0. This means that when two different elements are compared, the value for this measurement should be greater than 0. However, this would

imply that the value between two different elements is higher than the value when an element is compared to itself. This, of course, is in conflict with the idea that an increasing value will mean that two elements are more alike.

To conclude, there are three types of measurements to compare elements in a data set: distances, similarities and dissimilarities. The hybrid similarity can be used on all three kinds. Since it is calculated as the average of the original measurements, the value of the hybrid similarity will remain in the same range as the original measurement used by the data mining tool.

The downside of the hybrid similarity measure is that when it is used as a plugin for a data mining tool that uses a distance, the triangle inequality that held for the original distance does not necessarily hold for the hybrid similarity measure. This can easily be seen. Consider a data set $D = (V, E)$, with $V = \{u, v, w\}$ and $E = \{(u, v), (v, u), (u, w), (w, u)\}$. There also is a distance measure that gives 0 when the two elements are the same and 1 otherwise. Calculating the *contextual distances*, d_c, using (3.3) from Section 3.4.2, gives: $d_c(u, v) = 1/4$, $d_c(u, w) = 1/4$, and $d_c(v, w) = 1$. In the triangle inequality, $d_c(v, u)+d_c(u, w) \geq d_c(v, w)$, this will lead to $1/4 + 1/4 \geq 1$, which of course is not true.

This means that data mining tools that use the triangle inequality cannot use the hybrid similarity as such. This does not necessarily need to be a problem. Data mining tools that use a distance measure can be divided into three categories with regard to the triangle inequality:

1. The triangle inequality is not used in the data mining tool.

2. The triangle inequality is only used to speed up the data mining tool.

3. The triangle inequality plays an important role in the data mining tool.

It is obvious that for the first category, there is no problem in using the hybrid similarity measure. For the second category, the speed-up cannot be used, but the original algorithm should have no problem in using the hybrid similarity measure. Only for the third category is it impossible to use the hybrid similarity measure. Research should try to find out to what extent this actually is a problem.

4.4 Semantic Similarity v. Measured Similarity

A similarity measure is a tool that quantifies the degree to which two elements are similar, using the information about these elements in the dataset. The question is: to what extent does this calculated similarity relate to the actual, or semantic similarity between the two elements? Many different aspects can cause a difference between the semantic similarity and the calculated similarity. To get a good understanding of this, it is useful to first take a closer look at what this “semantic similarity” actually is.

The semantic similarity between two elements depends on the goals of the user. Consider a dataset with animals and their characteristics. If the user wants to divide these animals in classes like 'mammals,' 'birds,' 'fishes,' only some rough characteristics of the animals are needed. For instance, when two animals both have a spine, breathe air, have hair on their body, and breast-feed their young offspring, they can be considered similar, namely mammals. Another dataset with characteristics of animals may be used by park rangers to keep track of the movement of the more special animals (e.g. lions, rhinos, and elephants). In this case, to conclude whether two animals from different sightings are the same, it is necessary to look at more detailed characteristics such as individual markings or scars.

The semantic similarity between two elements also depends on the context: the other elements in the data set. Consider a dataset with texts where the user wants to group the texts that are about the same subject. When there are two texts about sport, whether or not they should be in the same group depends on the subjects of the other texts in this dataset. If the dataset consists of texts from different categories such as 'sport,' 'science,' 'romance,' or 'politics,' two texts about sport should be considered similar (i.e. about the same subject) and grouped together. On the other hand, if all the texts in the dataset are about sport, then categories can be 'football,' 'rowing,' or 'cycling,' or maybe those categories are 'professional,' 'recreational,' 'tactics,' or 'training,'.

In the previous cases, the semantic similarity was either TRUE or FALSE. Two animals are the same, or they are not the same; two texts are about the same subject, or not. In other occassions, the semantic similarity can be on a more gradual scale. Consider once again the dataset with texts about 'sport,' 'science,' 'romance,' 'politics,' and so on. When a text is about the love between two athletes, its subject is somewhere between 'romance' and 'sport,' making it difficult to determine exactly in which category this text belongs.

To conclude, the semantic similarity can be defined as "the actual similarity, the goal for the similarity measure, depending on the task and the context." Still, this definition is not very precise. In actuality, it is impossible to give a precise definition, because the semantical similarity is a very abstract concept. As a result, it is not straightforward how a similarity measure that calculates a value describing this semantical similarity needs to be defined. Besides that, there are other difficulties that arise when constructing a similarity measure, and which can cause a difference between the calculated similarity and the semantic similarity.

First of all, as described in Section 4.2, there is the principle of content variability. This states that the contents of elements from the same class vary around a center. A similarity measure uses this content to calculate how similar two elements are. So, when the contents of all elements in the same class vary and these contents are used to calculate the similarity, the similarities from one element to all elements in the same class also will vary.

Another problem is that it is sometimes difficult for a computer to get the semantic meaning of an element. Consider again a dataset with texts. For a human it is quite easy to understand what the subject of a text is, but for a computer this is much more difficult. One method to calculate the similarity between two texts is to calculate the cosine of the angle between their 'keyword vectors.' A keyword vector is a vector where each element of the vector represents a word that appears in any of the texts. The value for such an element is the amount of appearances of that specific word in that specific text. Two texts about the same subject have relatively more words in common, and thus the cosine of the angle between their keyword vectors will be higher.

The problem that arises is that the meaning of a word depends on the context. Consider, for instance the next two sentences: "A pitcher ducks when a bat flies to him." and "Ducks, flies and a bat drank from the pitcher." These sentence have relatively many words in common, but their meanings are very different. The opposite can also be true. Consider the sentences "I made an appointment with her by telephone" and "I called her so we could meet." These sentences have hardly any words in common, but mean roughly the same.

All in all, there is something like a "semantic similarity," which is the true similarity, and the goal of the measured similarity is to approach this semantic similarity as well as possible. This is difficult because the semantic similarity also depends on the goal of the user and the other elements in the dataset. Furthermore, grasping the semantic similarity in a numerical value could be difficult anyway. This difficulty arises because the contents of elements from the same class may vary around a mean, resulting in difficulties for a computer to get the semantic meaning of an element.

4.5 Hybrid Similarity v. Original Similarity

Following the definitions in Section 3.3, the dataset $D = (V, E, \alpha, \lambda)$ consists of V, a set of elements; E, a set of edges connecting these elements; α, a function that assigns an annotation to each element; and λ, a function that assigns a label to each element. The space where the semantic, or real, objects live will be called $\mathcal{R}$. An element $v \in \mathcal{R}$ is represented using an annotation $a \in \mathcal{A}$. We distinguish between the semantic similarity $\mathcal{S}_\mathcal{R} : \mathcal{R} \times \mathcal{R} \to \mathbb{R}$, and the similarity measure used by the DM tool $\mathcal{S}_\mathcal{A} : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$.

There is a difference between the real similarity $\mathcal{S}_\mathcal{R}$ and the measured similarity $\mathcal{S}_\mathcal{A}$. This difference can be considered 'noise.' Noise can arise in a dataset because it is difficult for a computer program to exactly get the semantic meaning of an element. This has been discussed previously in this chapter. Another possibility for noise to appear in a dataset is by errors that occur while processing the data.

Users of DM tools are interested in the real similarities, but since these similarities cannot be formalised, the measured similarity needs to be used.

Thus, it is important that the difference between $\mathcal{S}_{\mathcal{R}}$ and $\mathcal{S}_{\mathcal{A}}$ is as small as possible. This could be done by altering $\mathcal{S}_{\mathcal{A}}$ to make it more befitting $\mathcal{S}_{\mathcal{R}}$, but when this is not possible, other methods should be used to reduce the gap between these similarities. The hybrid similarity measures as described in Chapter 3 can do such a thing. To show this, we compare the content-based similarity with the neighborhood similarity as described in (3.2). To emphasize that they are both similarities that are calculated using the annotations, from now on they will be called $\mathcal{S}_{\mathcal{A}1}$ and $\mathcal{S}_{\mathcal{A}2}$, respectively.

The content-based similarity $\mathcal{S}_{content}$ is an estimator of the real similarity. Let ϵ denote the difference between this estimator and what it estimates. We then have

$$\mathcal{S}_{\mathcal{A}1}(v,w) = \mathcal{S}_{content}(v,w) = \mathcal{S}_{\mathcal{R}}(v,w) + \epsilon(v,w). \tag{4.1}$$

Any distribution over pairs of annotated nodes gives rise to distributions of $\mathcal{S}_{\mathcal{R}}$, $\mathcal{S}_{\mathcal{A}}$ and ϵ. If the estimator is unbiased, ϵ has a distribution centered around 0, so the expected outcome will be 0: $\mathbb{E}[\epsilon(v,w)] = 0$. Note that any estimator can be made unbiased by subtracting its bias from it. Therefore, in the following, we simply will assume that the estimator is unbiased. Hence $\mathbb{E}[\epsilon(v,w)] = 0$. Finally, we denote the variance of this distribution by σ_ϵ^2.

Now consider, as a second estimator, the neighbor similarity:

$$\begin{aligned} \mathcal{S}_{\mathcal{A}2}(v,w) = S_{neighbor}(v,w) &= \frac{\sum_{u\in\mathcal{N}(w)} \mathcal{S}_{content}(v,u)}{|\mathcal{N}(w)|} \\ &= \frac{\sum_{u\in\mathcal{N}(w)} \mathcal{S}_{\mathcal{R}}(v,u)}{|\mathcal{N}(w)|} + \frac{\sum_{u\in\mathcal{N}(w)} \epsilon(v,u)}{|\mathcal{N}(w)|} \end{aligned} \tag{4.2}$$

From this, consider the conditional distribution of $\mathcal{S}_{\mathcal{R}}(v,u)$, with $u \in \mathcal{N}(w)$ given $\mathcal{S}_{\mathcal{R}}(v,w)$. The semantic similarity $\mathcal{S}_{\mathcal{R}}(v,u)$ is likely dependent on $\mathcal{S}_{\mathcal{R}}(v,w)$ since w and u are connected. More specifically, due to homophily and transitivity, we expect them to be positively correlated. Let us write for $u \in \mathcal{N}(w)$:

$$\mathcal{S}_{\mathcal{R}}(v,u) = \mathcal{S}_{\mathcal{R}}(v,w) + \xi(w,u) \tag{4.3}$$

where ξ denotes the difference between the real similarity between v and w on the one hand, and the real similarity beween v and u on the other hand. This formula is generally valid (by definition of ξ), but in the presence of homophily and transitivity, we additionally expect ξ to be small.

Using (4.3) to substitute $\mathcal{S}_{\mathcal{R}}(v,u)$ in (4.2) gives

$$\mathcal{S}_{\mathcal{A}2}(v,w) = \mathcal{S}_{\mathcal{R}}(v,w) + \frac{\sum_{u\in\mathcal{N}(w)} \xi(w,u)}{|\mathcal{N}(w)|} + \frac{\sum_{u\in\mathcal{N}(w)} \epsilon(v,u)}{|\mathcal{N}(w)|}. \tag{4.4}$$

Observe that, among v, w and all $u \in \mathcal{N}(w)$, there is no relation except for the fact that w is connected to each element from $\mathcal{N}(w)$. This entails the

following. For symmetry reasons, the distribution of $\epsilon(v,u)$ must be equal to that of $\epsilon(v,w)$ (from v's point of view, u is just a random vertex, just like w). For each u, $\epsilon(v,u)$ therefore has an expected value of 0 and a variance of σ_ϵ^2.

Again because of symmetry, there is no reason to believe that, on average, $\mathcal{S}_\mathcal{R}(v,w)$ should be greater or smaller than $\mathcal{S}_\mathcal{R}(v,u)$. This means that $\mathbb{E}[\xi(w,u)] = 0$. Since all $u \in \mathcal{N}(w)$ are also interchangeable among themselves, all $\xi(w,u)$ have the same distribution. We denote the variances of these as σ_ξ^2.

Now consider the case where all $\epsilon(v,u)$ are independent, all $\xi(w,u)$ are independent, and $\epsilon(v,u)$ is independent from $\xi(w,u)$ for all $u \in \mathcal{N}(w)$. For this special case, we obtain the following squared errors for the two estimators:

$$SE(\mathcal{S}_{\mathcal{A}1}(v,w)) = \mathbb{E}[(\mathcal{S}_{\mathcal{A}1}(v,w) - \mathcal{S}_\mathcal{R}(v,w))^2] = \sigma_\epsilon^2 \tag{4.5}$$

$$SE(\mathcal{S}_{\mathcal{A}2}(v,w)) = \mathbb{E}[(\mathcal{S}_{\mathcal{A}2}(v,w) - \mathcal{S}_\mathcal{R}(v,w))^2] = \frac{\sigma_\epsilon^2 + \sigma_\xi^2}{|\mathcal{N}(w)|} \tag{4.6}$$

It is now obvious that the first estimator has an expected squared error of σ_ϵ^2, and the second estimator has an expected squared error of $(\sigma_\epsilon^2 + \sigma_\xi^2)/|\mathcal{N}(w)|$. This second term can be larger or smaller than the first. However, it tends to become smaller as $|\mathcal{N}(w)|$, the number of neighbors, increases. When (and whether) it becomes smaller than σ_ϵ^2 depends on the relative size of σ_ξ^2.

Note that, intuitively, σ_ϵ^2 is related to content variability (it reflects the extent to which the observed content similarity between two nodes approximates their real similarity), whereas σ_ξ^2 is related to the amount of homophily and transitivity in the network (it expresses to what extent nodes that are linked together have comparable similarities to other nodes). In networks with strong homophily and high content variability, the second estimator can be expected to be more accurate than the first.

The case where ϵ and ξ are not independent is mathematically more complex. We already have σ_ϵ^2 and σ_ξ^2 for the variance of $\epsilon(v,u)$ and $\xi(w,u)$. Now for every $u_i, u_j \in \mathcal{N}(w), u_i \neq u_j$, we denote the covariance between $\epsilon(v,u_i)$ and $\epsilon(v,u_j)$ as $\textsc{Cov}_\epsilon$, the covariance between $\xi(w,u_i)$ and $\xi(w,u_j)$ as $\textsc{Cov}_\xi$, and the covariance between $\epsilon(v,u_j)$ and $\xi(w,u_i)$ as $\textsc{Cov}_{\epsilon\xi}$. Then, using the rule that $\textsc{Var}(\sum_i X_i) = \sum_i \textsc{Var}(X_i) + \sum_{i,j\neq i} \textsc{Cov}(X_i, X_j)$, and exploiting the linearity of $\textsc{Var}$ and $\textsc{Cov}$, one quickly arrives at

$$SE(\mathcal{S}_{\mathcal{A}2}(v,w)) = \\ \frac{\sigma_\epsilon^2}{|\mathcal{N}(w)|} + \frac{\sigma_\xi^2}{|\mathcal{N}(w)|} + \frac{|\mathcal{N}(w)|-1}{|\mathcal{N}(w)|}\textsc{Cov}_\epsilon + \frac{|\mathcal{N}(w)|-1}{|\mathcal{N}(w)|}\textsc{Cov}_\xi + \textsc{Cov}_{\epsilon\xi} \tag{4.7}$$

which shows that strong positive autocorrelations of ξ and ϵ are likely to spoil the advantage of using $\mathcal{S}_{\mathcal{A}2}$. Such correlations are not impossible. For instance, $\epsilon(v,u_1)$ may be high because $\alpha(v)$ deviates strongly from its expected value (due to content variability), which makes $\epsilon(v,u_2)$ more likely to be high too. It is difficult to estimate how large this effect can be. On the other hand, if

$\epsilon(v, u_1)$ is high because $\alpha(v)$ deviates strongly from its expected value, it means that the content-based similarity as an estimator is very bad and there is much room for improvement. Also, this means that there is a good chance that, due to homophily and transitivity, the average similarity of w with all elements in the neighborhood of v is a much better estimator for the semantic similarity. Since this average is defined as $\mathcal{S}_{neighbor}(w, v)$, which is also incorporated in $\mathcal{S}_{contextual}$, it can be expected that the contextual similarity will be a better estimator for the semantic similarity.

Finally, note that $S_{context}(v, w)$ is the average value of the two similarities $S_{neighbor}(v, w)$ and $S_{neighbor}(w, v)$. Hence, it is likely to be slightly more accurate than $\mathcal{S}_{\mathcal{A}2}$ (its standard error is $\sqrt{2}$ times smaller, in case of independence of the error terms for v and w). This is again due to the error-reducing effect of averaging. The similarity $S_{combined}$, being the average of the similarities $S_{content}$ and $S_{context}$, can be more accurate than either or them, or have an accuracy somewhere in between.

4.6 Summary

Many data mining tools use a measurement to compare different elements in a data set. Three main categories for these measurements can be distinguished: distances, similarities and dissimilarities. It is difficult for a computer program to correctly calculate the real, semantic similarity between two elements. This will result in a difference between the semantic similarity and the measured similarity. There are several different causes for this:

- The semantic similarity can depend on the goals of the user.
- The semantic similarity can depend on the context in the data set.
- The semantic similarity can be on a gradual scale between two categories.
- Elements in the same class have a variety of content.
- The meaning of a single element can be difficult for a computer already.

All these problems make it difficult for a similarity measure to calculate the similarity between elements correctly. The hybrid similarity measure proposed in this thesis tries to improve the quality of the previously existing similarity measures by exploring the environment of an element. Due to homophily, it can be expected that the environment of an element tends to consist of elements of the same class. By taking the average similarity of the environment, problems that occur during the calculation of the similarity of a single element can average out, resulting in a more accurate similarity.

Part II

Implementations

Chapter 5

Agglomerative Hierarchical Clustering

Agglomerative hierarchical clustering is a method that forms clusters by merging the most similar existing clusters. The original similarity that this algorithm uses can be replaced by the hybrid similarity measures to include relational information. Experiments with both the original similarity and the hybrid similarities are done on the Cora dataset, a dataset with scientific papers. This should give a first indication whether using the hybrid similarities could lead to better results.

5.1 Introduction

In Chapter 2, it was explained that Knowledge Discovery in Databases (KDD) deals with the increasing demand of retrieving useful knowledge from rapidly growing databases. The most important step in the KDD-process is called Data Mining, and that is where the algorithm actually is doing its task. A data mining algorithm depends, amongst others, on the type of data it gets as input. One of the challenges of KDD is to address the problem that there are many different types of data to create algorithms for. It is even possible to combine different types of information in the same database, thereby creating a wide variety of different types of databases.

It is not feasible to create a different algorithm for every new combination of data types. Rather than doing that, it could be more beneficial to create a way to incorporate one type of information into a data mining algorithm that is created for another type of information. The method proposed in this thesis, the hybrid similarity measure, does exactly that.

Chapter 3 explained precisely the working of this method. The chapter showed that the hybrid similarity measure alters a similarity measure that is

based purely on the content of the elements in the database. It does so by regarding the neighborhood of the elements and the content-based similarities to them. The average of these similarities is treated as a new similarity and replaces the old similarity. Indirectly, the relational information is thus incorporated into a data mining algorithm that uses only content-based information.

One of the benefits of this hybrid similarity measure is that it is not necessary to invent new data mining algorithms for data that has both content-based and relational information. Other benefits were described in Chapter 4, where it was shown that there are always different types of noise or data variance in natural data sets, amongst others because of content variability. The hybrid similarity uses homophily and transitivity of similarity to make the data mining algorithm more robust to various types of noise.

In this chapter, the hybrid similarity measure will be implemented in an already existing data mining algorithm and used on real data to compare it with the original similarity. The data come from the Cora data set, which is presented in Section 5.2. The algorithm used is agglomerative hierarchical clustering, which is presented in Section 5.3. Section 5.4 presents the experimental setup in detail, after which Section 5.5 presents the results from the experiments. Finally, Section 5.6 presents the conclusions that can be drawn from these first results.

5.2 The Used Dataset

5.2.1 The Cora Dataset

The experiments are done on the Cora dataset, which was created by McCallum et al. [65]. They researched the possibility to use several machine learning techniques to automatically create a domain specific internet portal. Such a portal is an information gateway that often includes a search engine plus additional content. It enables the user to ask very specific, highly detailed queries on a certain subject.

Their process consisted of three steps. First, a web crawler searched for documents to add to the collection, using a reinforcement learning framework to decide which documents to add (see the survey by Kaelbling et al. [42]). After that, characteristic pieces of information were extracted from the documents using hidden Markov models (see the tutorial by Rabiner [78]). Finally, the documents were arranged into a hierarchical classification by means of an extended version of naive Bayes (see Lewis [58] or McCallum and Nigam [64]).

McCallum et al. demonstrated the working of these techniques by creating a portal for computer science research papers called "Cora." More than 50,000 papers are included in the database and available for keyword search. For 37,000 of these papers, the abstracts are also available. Besides that, all papers are placed in a computer science topic hierarchy with 70 classes at the leaves.

Finally, the citation links between these papers are available creating a big citation graph. This graph consists of many more papers, since it also includes all papers that are cited in the papers from the database, while the keywords or abstracts of these papers are not yet available.

The downside of this dataset is that it may not necessarily be correct. Since it has been created automatically, papers can be assigned to the wrong class. This makes it difficult to evaluate the result of a data mining algorithm. The fact that things have gone wrong, stems from the fact that some papers have more than one abstract or class assigned to it. From this, it can be assumed that more errors are made, but it is not possible to know how many, and, more importantly, which. For the purpose of evaluating data mining algorithms, this is not a big problem. The "ground truth" is defined as the information that is in the database. This is the reference point that algorithms are compared to.

5.2.2 The Created Subsets

The Cora dataset is created with a web crawler. Generally, a web crawler starts with one paper and from there tries to expand its scope by adding papers. Such a web crawler is sometimes called a spider, since it creates its web by starting in the center and then expanding it. When the crawler stops, the neighborhoods of elements in the central area are added to the dataset completely. In the same time, the neighborhoods of the elements from the areas on the edge (the relatively new papers) are not yet completely added to the dataset.

This means, of course, that the older regions of the dataset are better usable for data mining tasks than the newly added regions, which are yet to be fully processed. Unfortunately, it is difficult to determine which regions are complete and which are not yet. However, the connectivity of a particular region could show its completeness. For regions that have low connectivity, it could be assumed that not all the links that are there in the real world are already established for the dataset.

In an attempt to solve this problem, for this experiment, several subsets of the Cora dataset were created. All classes were sorted on connectivity. Here, not only the ratio between edges and vertices was taken into account, but also things like the amount of vertices that are connected to only one other vertex, and other comparable attributes are considered. Then, a first subset was created, named Cora-1, with the eight best connected classes. This dataset was extended four times by adding more classes. An overview of this can be seen in Table 5.1.

In this table, it can be seen that the average degree rises despite the fact that the connectivity of the classes that are added is supposed to be less than the classes that are already in a smaller subset. This can be explained by the edges that connect vertices from classes that were already in the subset with edges from classes that are newly added. In relation to the amount of vertices,

dataset	number of classes	number of papers	number of edges	average degree	graph density
CORA-1	8	720	2244	6.23	0.0087
CORA-2	17	1725	6093	7.06	0.0041
CORA-3	24	2523	10022	7.94	0.0032
CORA-4	31	4692	18634	7.94	0.0017
CORA-5	45	7917	39317	9.93	0.0013

Table 5.1: Overview of the five subsets from the Cora dataset and some of their characteristics. The 'average degree' is calculated as $\frac{2|E|}{|V|}$ and the 'graph density' is calculated with use of the definition by Coleman and Moré: $\frac{2 \cdot |E|}{|V| \cdot (|V|-1)}$ [14]

.

relatively more edges are added to a subset. However, the table shows that the overall graph density decreases. This becomes clear by using the definition of Coleman and Moré: $\frac{2 \cdot |E|}{|V| \cdot (|V|-1)}$ [14], which is basically the ratio between the edges that are in the graph and all the edges that could be in the graph (e.g. the amount of edges a complete graph with n vertices has).

5.3 The Method

One of the simplest ways of clustering elements is by using agglomerative hierarchical clustering. While the field of taxonomy already used similar techniques for a longer time, agglomerative hierarchical clustering was formalized by Johnson in 1967 [41]. For clustering n elements, it only needs a $n \times n$ similarity matrix S where an element s_{ij} denotes the similarity between elements v_i and v_j. Information about the elements is not necessary. It works as follows:

1. Assign every element to its own cluster.
2. Determine which clusters are most similar.
3. Merge these two clusters to a single cluster.
4. Update all similarities from this new cluster to the others.
5. Repeat steps 2 through 4 until all elements are in the same cluster.

The result of the algorithm is not a single, best clustering, but rather a hierarchy of found clusters. Normally, this hierarchy is represented in a *dendrogram*, which is a tree where all the leaves are the single elements and the root is the final cluster with all elements in it. The nodes in between represent one of the merges from step 3. Every level of this dendrogram can be considered a found clustering. An example of such a clustering is shown in Figure 5.1.

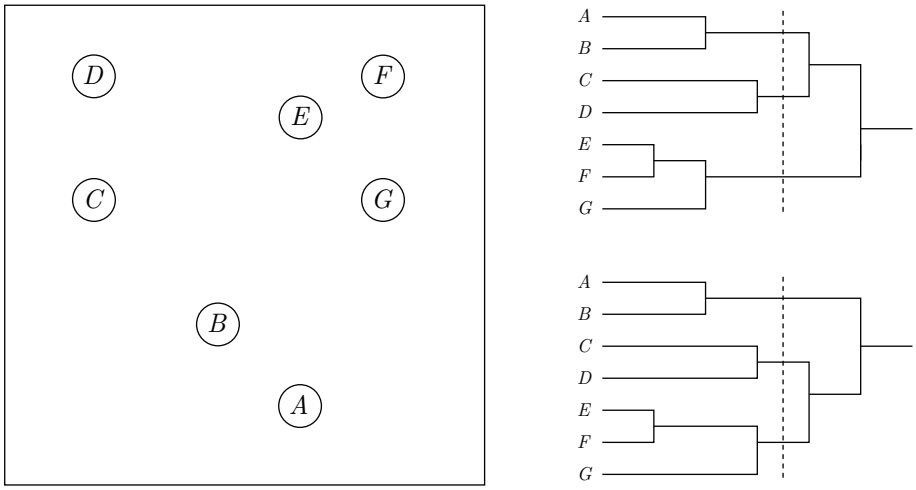


Figure 5.1: On the left are some elements that will be clustered with agglomerative hierarchical clustering. Here, instead of a similarity a distance measure is used. The distance between two elements is the physical distance in the drawing. On the right, two possible dendrograms are drawn. The top one is created by using single linkage and the bottom one is created with complete linkage. Both are created from left to right and the horizontal position shows when two clusters are merged during the process. At every position, one can cut the dendrogram to see a possible clustering. The dotted line is an example of such a cut. In both cases, it is of the clustering $\{\{A, B\}, \{C, D\}, \{E, F, G\}\}$. Before this dotted line, both methods produced the same clusters, but not necessarily in the same order. After the dotted line, the results start to differ. With single linkage, $\{A, B, C, D\}$ is formed, while with complete linkage, $\{C, D, E, F, G\}$ is formed. Both clusters do not appear anywhere in the other process.

The precise working of the algorithm, and the dendrograms resulting from that, depend for a big part on the way that the similarities are updated in step 4. There are several ways of doing that, each with their own benefits and downsides. The three most used methods are single linkage, complete linkage and average linkage.

Single linkage was originally defined by Florek et al. [26], and later reinvented by McQuitty [66] and Sneath [86]. In *single linkage*, the similarity between two clusters is the similarity of the two most similar of their elements that each come from a different cluster. More formally, given elements a and b, and clusters A and B, with $a \in A$ and $b \in B$, then the similarity between A and B, $\mathcal{S}(A, B)$, is defined by $max\{\mathcal{S}(a, b)\}$.

Complete linkage originally was defined by Sørensen [89]. It can be considered the opposite of single linkage. With *complete linkage*, the similarity between two cluster is the similarity of the two least similar of their elements that each come from a different cluster. Formalized with the same elements as before, $\mathcal{S}(A, B) = min\{\mathcal{S}(a, b)\}$.

Trying to strike the golden mean between single linkage and complete linkage, Sokal and Michener [88] came up with *average linkage*, where the similarity between two clusters is defined as the average of all similarities between an element from one cluster and an element from the other. It is formalized with the same elements as previously used: $\mathcal{S}(A, B) = \frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} \mathcal{S}(a, b)$.

The clustering hierarchies found can be very different when a different linkage method is used. Figure 5.1 shows an example of different results depending on the linkage method. Each method has its own good and bad sides. In an extensive study, Milligan compared several linkage methods [67]. The problem of single linkage is that there is a great chance of chaining. This means that elements are added to a cluster one at a time, because they are similar to the most recently added element. This is done without taking into account the similarity compared to the other elements in that cluster. The result can be long chains of elements that are all in the same cluster, but are spread throughout the entire domain. This problem is known as *chaining*.

The problem of chaining is solved by using complete linkage. Complete linkage has the tendency to form clusters of roughly the same diameter, and thus, the problem of chaining does not appear with complete linkage. Unfortunately, it can be heavily distorted by outliers. Since the distance is determined by the pair of least similar elements, it can be expected that at some point during the execution of the algorithm, the outliers often will be the elements determining the similarity between clusters.

Average clustering suffers much less from chaining than single linkage, and it is less sensitive to outliers than complete linkage. It is therefore understandable that, from these three linkage methods, it is the most popular. Still, there can be many situations where single or complete linkage can lead to better results than average linkage. There are also other linkage methods, like Ward's minimum-

variance method (that tries to minimize the variance between elements in a cluster [98]), and the centroid method by Sokal and Michener [88] (that uses the centroids of the clusters to calculate the similarities between them).

5.4 Experimental Setup

We briefly recall the formal definitons from Section 3.3. All experiments are done on a dataset $D = (V, E, \alpha, \lambda)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices or elements, $E \subseteq V \times V$ is the set of edges, $\alpha : V \to \mathcal{A}$ a function that assigns to any vertex v of V an "annotation," and $\lambda : V \to \mathcal{L}$ a function that assigns to any vertex v of V a "label." Also, the neighborhood of a vertex v is defined as $\mathcal{N}(v)$, where $\mathcal{N}(v)$ is the set of vertices that are connected to v with an edge. So with two vertices $v, w \in V$ this means that $w \in \mathcal{N}(v) \Leftrightarrow (v, w) \in E$.

In this case, the dataset is any of the five subsets taken from the Cora dataset, as described in Section 5.2.2. The vertices or elements in the dataset are the papers, and the edges are the citations. (i.e. the edge $(v, w) \in E$ if and only if v cites w or w cites v). The labels are the classes that were assigned to the papers by the creators of the Cora dataset.

The Annotations

The space of annotations was left open in Section 3.3, but now it needs to be well defined. The annotations are derived from the words that appear in the abstracts of the papers. For this, first, it needs to be determined what all the useful words in all the abstracts are. This is the set $P = \{p_1, p_2, \ldots, p_m\}$, and is created by taking every word that appears in at least two different abstracts. From this set P, a set of known stop words is removed. Stop words are words that are so general that they do not have any power to make a distinction between different texts. Examples of stop words are 'the', 'is, 'would', and so on. For Cora-1, this set P contains 2924 words. As the subsets grow bigger, so do the corresponding sets of P up to 9082 words for Cora-5.

Using this set of useful words P, the annotation a_i of a vertex v_i can now be defined as a vector $\vec{a_i} = (a_{i0}, a_{i1}, \ldots, a_{im})$ where a_{ij} is 1 if p_j appears in the abstract of v_i, and 0 otherwise. These vectors can be considered the content of the elements and are used to calculate the content-based similarity.

The Similarity Measures

The content-based similarity is defined solely on the annotations of the elements. Since these annotations are vectors that state what words appear in the abstract, a good similarity should return a high value for a similarity between two elements when they have many words in common. From a vector perspective, when two abstracts have relatively many words in common, their

annotation vectors will point roughly in the same direction, and thus, the angle between them is relatively small. Therefore, the cosine of the angle between two annotation vectors can be used very well as a similarity measure, as it gives 1 when they point in the exact same direction and 0 when they are perpendicular. Since, in this case, the elements of the vectors cannot have a negative value, the cosine cannot be lower than 0.

More formally, given two elements $v_i, v_j \in V$ and their annotations defined by the function α, so that $\alpha(v_i) = \vec{a_i}$ and $\alpha(v_j) = \vec{a_j}$, then the content-based similarity between v_i and v_j is defined by:

$$\mathcal{S}_{content}(v_i, v_j) = \cos \angle(\vec{a_i}, \vec{a_j}) = \frac{\vec{a_i} \cdot \vec{a_j}}{|\vec{a_i}| \cdot |\vec{a_j}|} \tag{5.1}$$

With the content-based similarity defined, the contextual similarity and the combined similarity can be calculated easily. Following the definitions from Section 3.4.2, the contextual similarity is defined by:

$$\mathcal{S}_{contextual}(v_i, v_j) = \frac{1}{2} \cdot \left(\frac{\sum_{u \in \mathcal{N}(w)} \mathcal{S}_{content}(v, u)}{|\mathcal{N}(w)|} + \frac{\sum_{u \in \mathcal{N}(v)} \mathcal{S}_{content}(w, u)}{|\mathcal{N}(v)|} \right) \tag{5.2}$$

Finally, the combined similarity is defined following the definition from Section 3.4.3:

$$\mathcal{S}_{combined}(v, w) = c \cdot \mathcal{S}_{content}(v, w) + (1 - c) \cdot \mathcal{S}_{context}(v, w) \tag{5.3}$$

with $0 \leq c \leq 1$. c determines the weight of the content-based similarity in the combined similarity. Since no significant difference was found in performance, c was chosen as $\frac{1}{2}$.

The Linkage Method

With these similarities, the elements in the dataset can be clustered. When doing so, a linkage method needs to be chosen. In this case that is average linkage. For the experiments in this chapter, the exact linkage method is not important. The goal of the experiments is not to find the best clustering, but to see whether the original method can be improved.

The Evaluation Method

To compare different clustering methods, an evaluation method is needed. For this, Larsen and Aone's F-score [56] was used. It is a method to compare one clustering to another. A higher F-score means that those two clusterings are more similar to each other than two clusterings that have a lower F-score. A score of 1 means they are completely identical. For the elements in the

Cora dataset, all the labels are known and also can be seen as a clustering. This clustering is, of course, the ultimate goal for a clustering algorithm. The higher the F-score between the found clustering and the clustering based on the labels, the better the algorithm works. Therefore, to facilitate the distinguishing between the two, a cluster in the clustering based on the labels will be called a label class.

To compare the found clustering with the classes, two aspects of F-score are important: precision and recall. Considering a cluster C from a clustering $\mathcal{C}$ and a label class L from the set of all labels $\mathcal{L}$, the *precision*, $P(C, L)$, is the amount of elements from label class L in cluster C, divided by the total amount of elements in C. The *recall*, $R(C, L)$, is the amount of elements from label class L in cluster C, divided by the total amount of elements in L. Therefore, every class-cluster combination has its own precision and recall. Now, the F-score can be calculated as:

$$F\text{-score}(C, L) = \frac{2 \cdot P(C, L) \cdot R(C, L)}{P(C, L) + R(C, L)} \tag{5.4}$$

This does not yet give the F-score for the entire clustering in comparison to the complete labeling. To do so, for every label class, the best fitting cluster needs to be found. This is the cluster that has the best F-score for that class. Then, of all these F-scores, the weighted average (with regard to the size of the label class) is taken. Thus, the F-score for a clustering $\mathcal{C}$ with regard to a labelling $\mathcal{L}$ can be calculated with:

$$F\text{-score}(\mathcal{C}, \mathcal{L}) = \frac{1}{n} \cdot \sum_{L \in \mathcal{L}} (|L| \cdot \max\{F\text{-score}(C, L) \; \forall C \in \mathcal{C}\}) \tag{5.5}$$

The agglomerative hierarchical clustering starts with a clustering that consists of n clusters containing one element, and then stepwise merges these clusters until only one cluster remains. Every step can be seen as a clustering and thus at every step, the found clustering can be compared with the classes. During these experiments, the F-score was calculated every 10 steps to monitor its development during the procedure. The results, when using the original, content-based similarity, can be compared with the new hybrid similarities using these F-scores.

5.5 Results

The main goal for these experiments is to compare the original, content-based similarity with the new hybrid similarities: the contextual similarity and the combined similarity. This can be done best by comparing the F-scores throughout the clustering process. They can be seen in Figures 5.2 and 5.3 for all the five subsets.

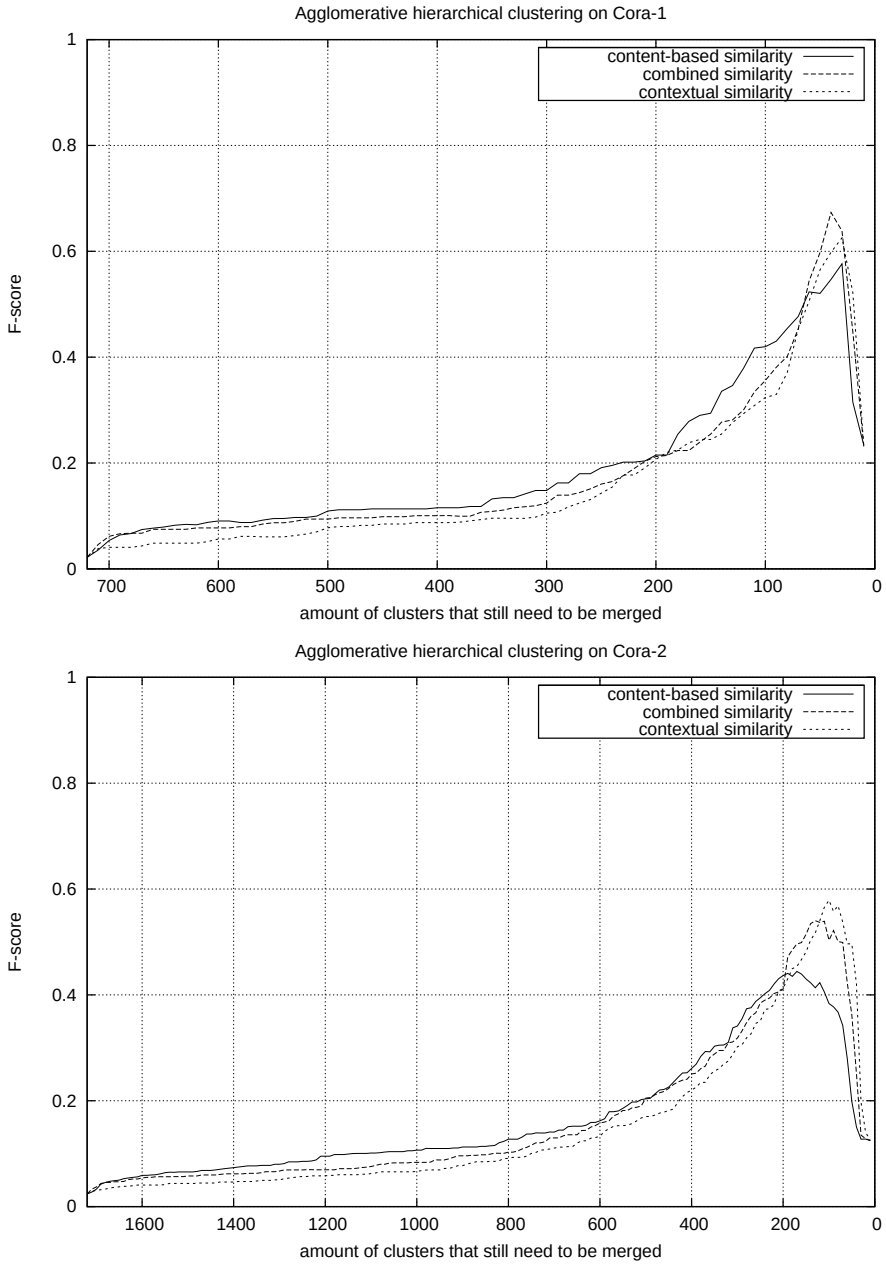


Figure 5.2: F-score for all the found clusterings during the process of agglomerative hierarchical clustering with average linkage on CORA-1 (top) and CORA-2 (bottom).

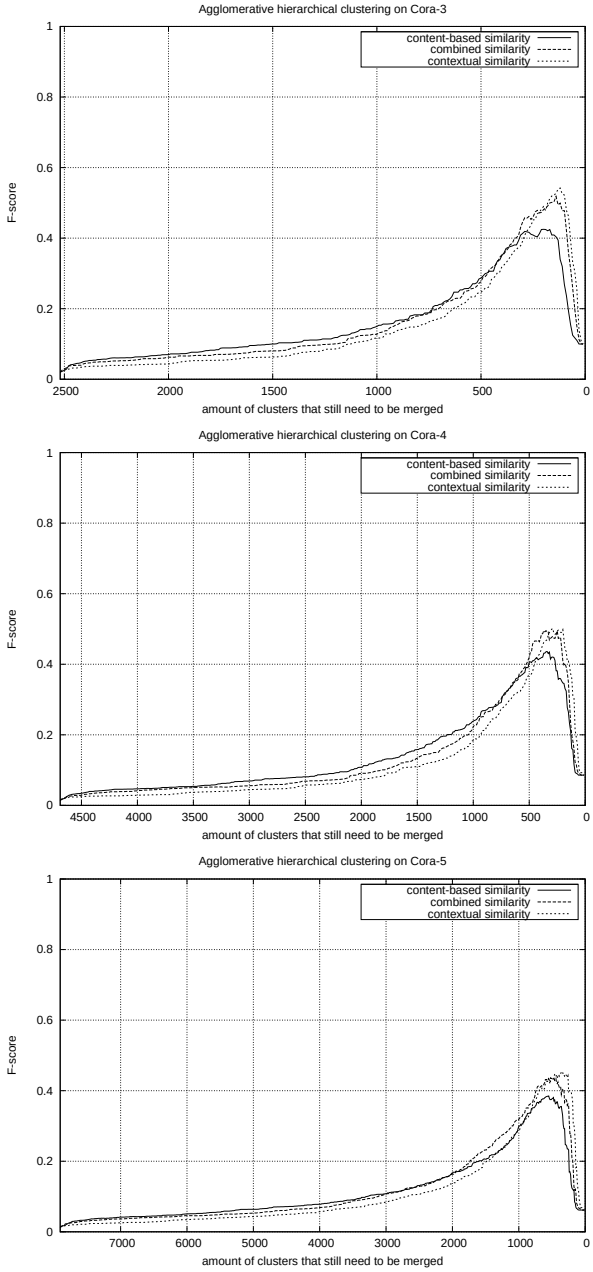


Figure 5.3: F-score for all the found clusterings during the process of agglomerative hierarchical clustering with average linkage on respectively CORA-3 (top), CORA-4 (middle), and CORA-5 (bottom).

Every line in these graphs shows the same characteristics; they start very low, rising slightly. As the clustering continues, the F-scores rise faster and faster, until a maximum is reached and the line drops rapidly. Also in all the graphs, the three similarities behave more or less in the same way with respect to each other. The content-based similarity performs better than the two hybrid similarities at first. Then, it reaches its maximum before the hybrid similarities do, and the F-score of the content-based similarity starts to drop. While this happens, the F-scores of the contextual and combined similarities continue to rise and they reach a significantly better score.

Another aspect that can be regarded is the found dendrogram. Figures 5.4 through 5.6 show these for respectively the content-based similarity, the combined similarity, and the contextual similarity. Due to lack of space, the complete dendrograms cannot be shown. The numbers on the leafs represent the amount of elements that are still on that branch. The main thing that can be noticed is that at the end of the clustering process, there are still many singletons that need to be merged. These are probably outliers that are difficult to cluster. Still, it looks like the hybrid similarities have a slightly better spread dendrogram than the content-based similarity. This could explain the results for the F-score.

5.6 Conclusion

In a relational context, one may want to cluster objects based on both their content and the relationships between them, with the latter being indicated by a graph. We have proposed a method for adapting the distance or similarity measure in such a way that relational information is inserted. Experiments on the Cora data set show that this more informed similarity measure leads to better clustering results when it is plugged into a standard clustering procedure.

It is difficult to say anything about the found F-scores in general. Since the used subsets are custom-made for these experiments, the results cannot be compared with results from other experiments. What can be said about the results however, is that the found dendrograms are unbalanced. As this is probably due to outliers, a better handling of these outliers might improve the F-scores even more.

However, the used similarities during these experiments can still be compared easily. A first conclusion can be that it seems that adding relational information does enhance the performance of this clustering procedure on these data sets. The content-based similarity outperforms the others from the start of the clustering for a long time, but the hybrid similarities overtake it when it matters most: when the amount of clusters starts to approach the amount of classes. This better performance can also be seen to some extent in the dendrograms for the hybrid similarities that appear to be slighty less unbalanced than that of the content-based similarity.

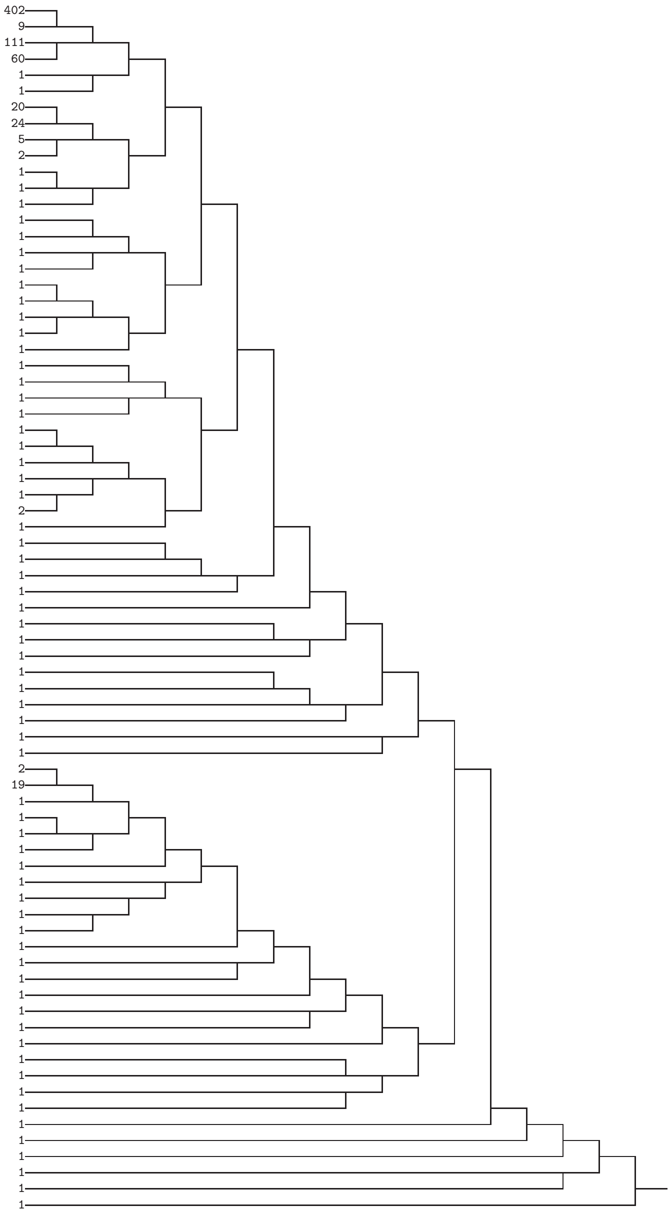


Figure 5.4: Dendrogram for the result on Cora-1 with the content-based similarity.

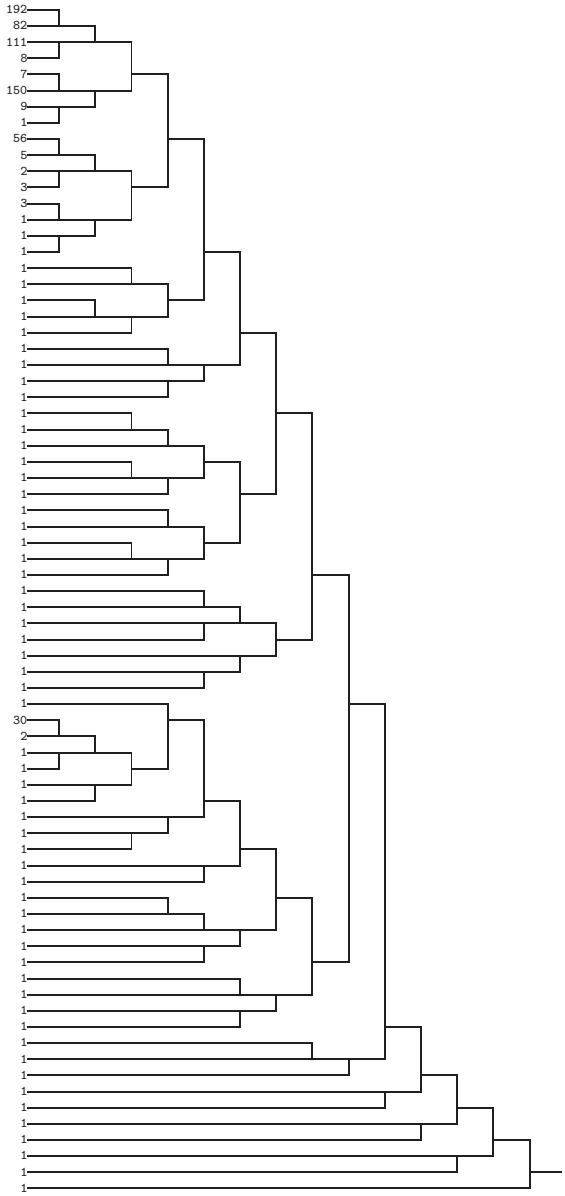


Figure 5.5: Dendrogram for the result on CORA-1 with the combined similarity.

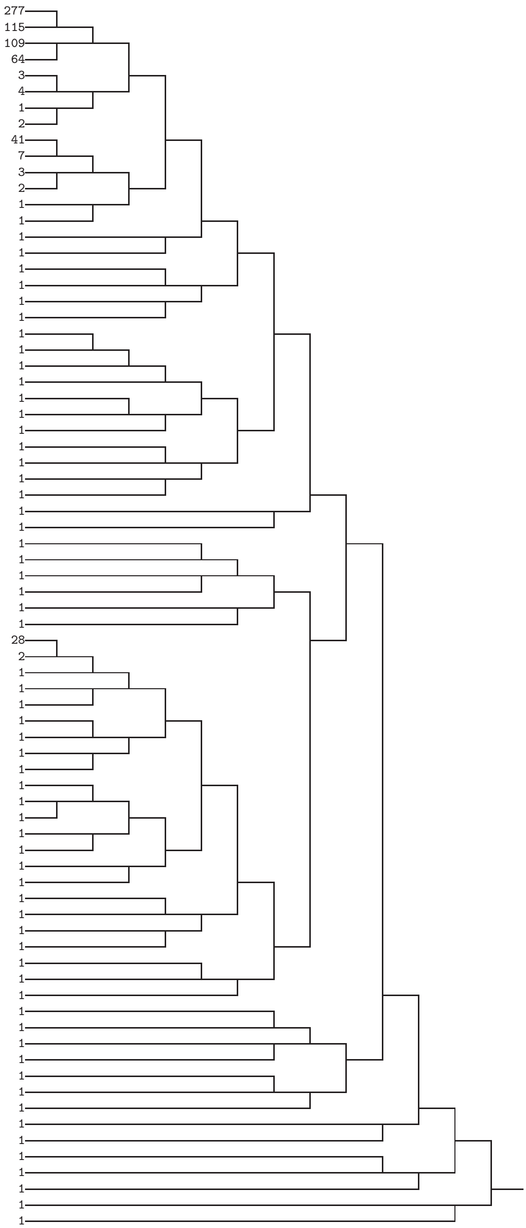


Figure 5.6: Dendrogram for the result on Cora-1 with the contextual similarity.

Chapter 6

K-means and K-medoids

Centroid-based clustering methods form clusters around a certain number of prototypes. Implementing the hybrid similarities in this type of clustering methods is not straightforward. It is not always possible to define the concept of a 'mean node' while at the same time having a meaningful hybrid similarity to it. Two ways around this problem are proposed. Experimental research will show both if there is a difference in performance for these two solutions and what the effect of using the hybrid similarities is.

6.1 Introduction

In previous chapters, we expressed the need for generative methods to create hybrid data mining techniques. We proposed two hybrid similarities, which can be used instead of the original, content-based similarity. The first experiments were set up to demonstrate the effect for agglomerative hierarchical clustering. Chapter 5 describes how this is done, and what the effects of using the hybrid similarities instead of the content-based similarities are on the quality of the found clusterings.

These experiments showed that the hybrid similarities performed significantly better than the content-based similarity. This leaves the question whether the hybrid similarities also will improve other data mining techniques. This chapter explores the possibilities to implement the hybrid similarities in two centroid-based clustering methods: k-means and k-medoids.

It will become clear that there are no obstacles in using the hybrid similarities with k-medoids; this is not the case for k-means. The hybrid similarities define a similarity between two elements in the dataset. However, k-means uses similarities between an element and a mean that is not part of the dataset. Since these kinds of similarities have not yet been defined, using the hybrid

similarity with k-means is not so straightforward. We propose two possible solutions to work around this problem.

The rest of this chapter is organized as follows. Section 6.2 explains the working of k-means and k-medoids. Section 6.3 describes how the hybrid similarities could be implemented for k-means and k-medoids. It also describes the problem that arises when trying to implement it in k-means, and proposes two ways around this problem. Section 6.4 describes the experimental setup and Section 6.5 describes the results. Finally, Section 6.6 draws conclusions from the found results.

6.2 The Methods

There are many different clustering methods. Amongst them are two important categories: hierarchical clustering and centroid clustering. Hierarchical clustering forms clusters by recursively splitting up or merging already existing clusters. Centroid clustering forms clusters around some centroids that are iteratively adapted to fit the actual clusters in the dataset more every step. The algorithms k-means and k-medoids are examples of centroid clustering.

6.2.1 K-means

Officially, k-means was created by MacQueen in 1967 [61]. He was the first to use the term 'k-means', although Steinhaus already mentioned this idea in 1957 [92]. Originally, the algorithm aims to cluster a dataset with elements that are vectors in an n-dimensional space. The Euclidean distance is used to calculate the distance between points in this n-dimensional space.

The algorithm starts by choosing k random points in the n-dimensional space. These can be considered the prototypes of the clusters. Then, the algorithm iteratively executes an assignment step, followed by an update step. During the assignment step, each element is assigned to the prototype that it is closest to (i.e. the prototype to which the Euclidean distance is the smallest). During the update step, for each cluster, the prototype is recalculated as the mean value of all elements in that cluster at that time. These steps are repeated until the means do not change anymore and all elements stay in the same cluster. More formally:

1. Randomly choose k different points M_i ($i = 1, \ldots, k$) in the data space; each M_i will serve as a prototype for a cluster C_i.
2. Assign each data element to the cluster with the prototype that has the lowest Euclidean distance to it.
3. Recalculate each M_i as the mean of all elements of C_i.
4. Repeat steps 2 and 3 until the M_i and C_i no longer change.

Here, step 2 is known as the *assignment step* and step 3 is known as the *update step*. An example of the working of this algorithm is shown in Figure 6.1.

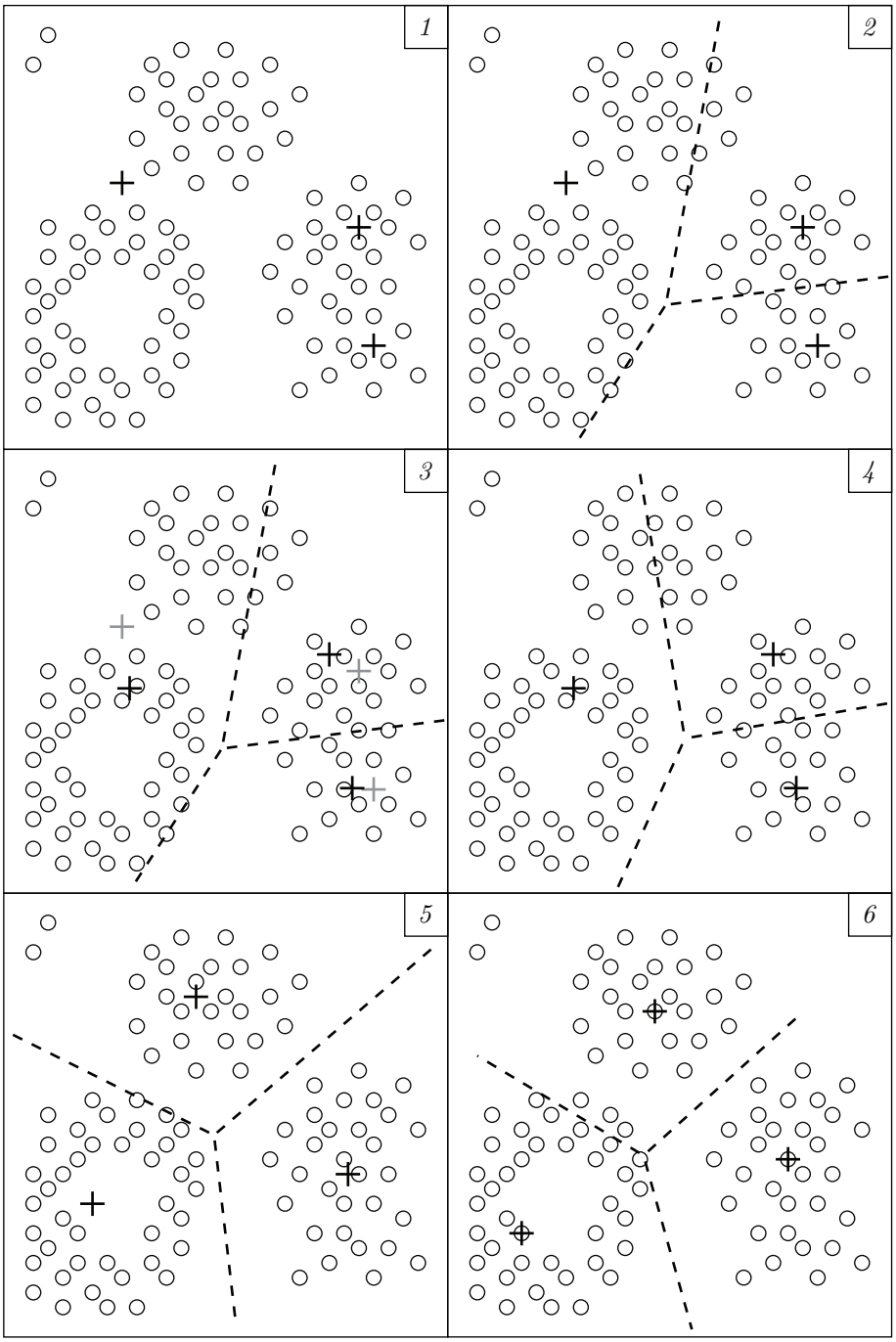
Although k-means always converges to an optimum, there is a chance that this is just a local optimum. The proof of this involves the fact that the sum of all squared Euclidean distances from one prototype to all the elements in that cluster can only decrease in each update and assignment step. Only a finite number of such decrements is possible (see, for instance, the book by Manning et al. [62]).

The size of k is left open. Much research has been done to determine the best value for k, given a specific dataset. One well-known method is the elbow method, which regards the percentage of variance explained by the clusters. This is the ratio of the between-group variance to the total variance. When the amount of clusters is increased, this percentage will begin to rise fast, meaning that an extra cluster will provide much extra information. At some point, the marginal gain will drop, leaving an elbow in the graph. This is the amount of clusters that should be chosen. The first one to come up with this idea was Thorndike in 1953 [95]. There are other methods, like, using information criteria (see, for instance, the work of Goute et al. [30]), or analyzing the kernel matrix (see, for instance, the work of Honarkhah and Caers [37]). If all methods fail, one can always use the rule of thumb to choose k as $\sqrt{\frac{n}{2}}$ (see the book by Mardia et al. [63]).

6.2.2 K-medoids

As a variant of k-means, Kaufman and Rousseeuw came up with k-medoids in 1987 [45]. It differs from k-means in that the prototype of a cluster (in this case known as the medoid) is always an element from the data set: in the beginning (step 1) it is a random data element; and during an update step (step 3), M_i becomes the element of C_i for which the overall similarity to all other elements of C_i is maximal.

Using k-medoids can be a good alternative for k-means. They both have their own benefits and downsides. For instance, k-medoids is known to be better than k-means when it comes to handling outliers, as shown by Han et al. [32]. This can be seen easily. Consider, for instance, an outlier v, which is assigned to a cluster C. The element v does not look like any of the other elements in this cluster. Nonetheless, it does not look like any of the elements in the dataset anyway. It still needs to be assigned to a cluster. Therefore, it was assigned to cluster C. When the prototype is calculated in the update step during k-means, the outlier v has a big influence on the mean value of all the elements in the cluster C. To make matters worse, the more v differs from the other elements in C, the bigger its influence is on the mean. This is illustrated in Panel 5 of Figure 6.1, where the mean of the top cluster is shifted a bit to the top left because of the outliers in that corner.


1
2
3
4
5
6

With k-medoids, during the update step, the medoid is chosen that has the highest overall similarity to all other elements in the cluster. The influence that v has on the sum of similarities for any possible medoid is small, and more importantly, it is roughly the same for any possible medoid in the cluster. If outliers have a big influence, then this is not good for the performance of an algorithm. Hence, k-medoids performs better in handling outliers than k-means.

A downside of k-medoids with respect to k-means, is that it is more limited in its choice of prototypes, as shown by Kirsten and Wrobel [48]. This can be a problem when clusters have a relatively sparse center, and are close to each other. In this case it is difficult for k-medoids to find the right medoid for those clusters. This is illustrated in Panel 6 of Figure 6.1. Here, the medoid of the bottom left cluster cannot be chosen somewhere in the center of the cluster. The effect is that some of the elements in this cluster are assigned to the top cluster.

Also, Han et al. showed that k-medoids is less efficient when handling big data sets [32]. To calculate the new prototype of a cluster in k-medoids, one needs to calculate the distance from every node in that cluster to every other node in that cluster. However, to calculate the new prototype of a cluster in k-means, one only needs to calculate the mean of all nodes in it.

6.3 Implementing the Hybrid Similarities

6.3.1 K-means and the Hybrid Similarities

K-means was designed for an n-dimensional space where Euclidean distances are used to determine which elements are close and which are far away. The properties of Euclidean space are used to prove that k-means always converges. This is an important aspect. Still, k-means has been used succesfully in other environments like radial basis function networks (Moody and Darken [69]), feature classifiers (Huang and Lippman [39]), and astronomy time series (Lei et al. [57]).

Figure 6.1 *(preceding page)*: Example of the working of k-means. Panel 1 shows the initial situation. Here, the circles are the elements that need to be clustered, and the crosshairs are the randomly chosen prototypes. It is clear to see that there are three clusters and two outliers, and that the initial prototypes are chosen poorly. Panel 2 shows that each element is assigned to the prototype it is closest to. Here, the exact center of a circle determines to which cluster it is assigned. In Panel 3, the prototypes are being updated, and shift towards the centers of the obvious clusters. In Panel 4, the elements are assigned again, but now to the updated prototypes. It is clear to see that the borders of the clusters shift to their desired locations. Panel 5 shows the end result for k-means. This can be compared with Panel 6 which shows the end result for k-medoids.

When k-means is implemented for a dataset that is an annotated graph, as defined in Section 3.3, it uses a prototype function $\mathcal{P} : 2^V \to \mathcal{A}$, that, given a subset of vertices, returns a prototype. Normally, this is the average of the annotations of the vertices in a cluster. Also, it needs a similarity function $\mathcal{S}' : \mathcal{A} \times V \to \mathbb{R}$, that calculates the similarity between a prototype and an element. The prototype function is used in the update step, and the similarity function is used in the assigment step.

There must be some aggregate function of the similarities between elements and prototypes (e.g. increasing sum of similarities) to guarantee convergence. During each assignment step and each update step, this function must increase (or decrease) monotonically. When this function does not increase (or decrease) anymore, then, and only then, the algorithm has reached its convergence. Thus, the prototype function and the similarity function must be compatible with each other (i.e. they both must have a similar effect on the aggregate function) to guarantee that the algorithm converges.

The prototype can be defined as an annotation that is not a part of the graph. However, the hybrid similarity measures are functions $\mathcal{S} : V \times V \to \mathbb{R}$, that define a similarity between two elements in a graph, and thus need two vertices of the graph as input. Fortunately, it is possible to rewrite the hybrid similarities in such way that they can be used to calculate the similarity between an element and a prototype. The problem is, that these similarity measures will not be compatible with the previously described prototype measure. We will discuss two ways around this problem. The first one will redefine the prototype measure in such a way that it will be compatible with the slightly altered versions of the hybrid similarity measures. The second one will be a more efficient approximation of the first. We next discuss these two methods.

K-means-NAM: K-means with Neighbor Annotation Means

Recall that the content-based similarity from (3.1) only uses the annotations of the elements. Since the prototype is in the same domain as the annotations, the content-based similarity between a prototype M and an element v can be defined as:

$$\mathcal{S}'_{content} : \mathcal{A} \times V \to \mathbb{R} : \mathcal{S}'_{content}(M, v) = \mathcal{S}(M, \alpha(v)) \tag{6.1}$$

The contextual similarity $S_{context}$ is a symmetrized version of $S_{neighbor}$. The definition of the latter (see (3.2)) uses for the first element (v) only its annotation $\alpha(v)$, not its location in the graph. Thus, the neighbor similarity $\mathcal{S}_{neighbor}$ can be rewritten as a function $\mathcal{S}'_{neighbor} : \mathcal{A} \times V \to \mathbb{R}$ that is defined by:

$$\mathcal{S}'_{neighbor}(M, v) = \frac{\sum_{w \in \mathcal{N}(v)} \mathcal{S}_{content}(M, w)}{|\mathcal{N}(v)|} \tag{6.2}$$

We can use this asymmetric neighbor similarity instead of the contextual similarity as described in (3.3). Also the combined similarity can then be rewritten as a function $\mathcal{S}'_{combined} : \mathcal{A} \times V \to \mathbb{R}$ that is defined by:

$$\mathcal{S}'_{combined}(M, v) = c \cdot \mathcal{S}_{content}(M, v) + (1 - c) \cdot \mathcal{S}'_{neighbor}(M, v) \tag{6.3}$$

Now that the hybrid similarities are rewritten, they can also calculate the similarity between an element and an annotation that is not part of the graph. Now, it is possible to calculate the prototype as the average of the annotations of all elements and still have a similarity measure that can be used to assign the elements to the most similar prototype.

This approach causes a new problem though: the proposed prototype measure and similarity measure are not compatible. In the update step, the new prototype is the mean of the cluster element's annotations, which increases the average content similarity between M and the nodes, but not necessarily the neighbor similarity between M and these nodes. As a result, k-means may no longer converge. Consider an element v that has an annotation that differs a lot from the annotations of its neighbors. When the contextual similarity is used, v will be placed in a cluster with a mean that is close to the annotations of the neighbors of v. However, when the mean for this cluster is updated, this will be done using the annotation of v, instead of the annotations of the neighbors of v. This will pull the mean towards v's own annotation, and away from the annotation of its neighbors. The effect could be that the new mean will be far from the annotations of v's neighbors, so the monotonic increase of the aggregate function is no longer guaranteed, and the algorithm may not converge.

To ensure convergence, we need to redefine the prototype measure to be compatible with the similarity measure. For this, we first need to limit the space of possible annotations a little more. From now on we assume that this space allows for a meaningful definition of the basic arithmetic operations: addition, subtraction, multiplication and division. As described in Section 3.3, $\alpha : V \to \mathcal{A}$ assigns an annotation to a vertex. Now let $\alpha' : V \to \mathcal{A}$ be a new function that assigns another annotation to a vertex:

$$\alpha'(v) = \frac{\sum_{w \in \mathcal{N}(V)} \alpha(w)}{|\mathcal{N}(v)|} \tag{6.4}$$

So $\alpha'(v)$ is the mean annotation of all neighbors of v. It is clear that calculating the prototype as the average of these new annotations is compatible with the proposed similarity measure.

Following the same reasoning, when using the combined similarity instead of the contextual one, the annotation function $\alpha'' : V \to \mathcal{A}$ should be used:

$$\alpha''(v) = \frac{\alpha(v) + \alpha'(v)}{2} \tag{6.5}$$

This setup, which we call *k-means-NAM* (*k*-means with Neighbor Annotation Means) is the first solution proposed to enable the use of a *k*-means-like algorithm with a hybrid similarity measure.

K-means-NAMA:
K-means-NAM Efficiently Approximated

The previous solution is less efficient than the original *k*-means algorithm. To calculate the similarity between a prototype and an element, the original *k*-means only needs to calculate one content-based similarity. The hybrid similarity for *k*-means-NAM, on the other hand, is based on the neighbor similarity. This similarity needs to calculate the content-based similarity between the prototype and all neighbors of that element. In every assignment step, this needs to be done for every prototype-element combination. Therefore, *k*-means-NAM is several times slower than the original *k*-means.

A more efficient alternative is to average the neighbor annotations themselves, instead of averaging the similarities. This would mean that the algorithm would use the content-based similarity on annotations that are adapted to indirectly include the relational information. Then, the formula to calculate this similarity would be:

$$\mathcal{S}_{content}\left(M, \frac{\sum_{w \in \mathcal{N}(v)} \alpha(w)}{|\mathcal{N}(v)|}\right) = \mathcal{S}_{content}(M, \alpha'(v)) \tag{6.6}$$

These two calculations are mathematically different, and generally do not give the same outcome, but they approximate each other well when the annotations of the neighbors ($\alpha(w)$) are "in the same direction" as $\alpha(v)$. The advantage of this additional approximation is that for each v, $\alpha'(v)$ can be computed once in advance, and substituted for $\alpha(v)$, after which the standard *k*-means algorithm can be run. This new annotation α'' approximates *k*-means-NAM with the combined distance in the same way that using α' instead of α allows us to approximate *k*-means-NAM with contextual distance. We call this approximation *k*-means-NAMA: *k*-means with Neighbor Annotation Means efficiently Approximated.

6.3.2 K-medoids and the Hybrid Similarities

It is obvious that the hybrid similarity measure can be applied to *k*-medoids without problems. Since the prototype is always an element in the data set (i.e. a node in the graph) the similarity with other elements can be calculated using the hybrid similarities from Section 3.4. To compute the new prototype, one only needs to compute for each element the sum of the similarities to all other elements in that cluster in order to determine which is the largest.

6.4 Experimental Setup

Three methods have been proposed: k-means-NAM, k-means-NAMA, and k-medoids. Each of these methods can be used with one of three similarity measures: the content-based similarity, the contextual similarity and the combined similarity. This leaves nine combinations of a method with a similarity measure. Two of these combinations are the same: k-means-NAM with the content-based similarity, and k-means-NAMA with the content-based similarity. They both are the original version of k-means that we aim to improve.

To evaluate the usefulness of the proposed methods, a few questions need to be answered experimentally:

1. For k-means, we have to choose between standard k-means with the content-based similarity, or an approximative variant that takes contextual information into account; does the use of contextual information compensate for a possible quality loss for having to use an approximate method?

2. When choosing such an approximate method, is there a difference in performance between k-means-NAM ans k-means-NAMA?

3. One can use k-medoids both with content-based or hybrid similarities; does the use of a hybrid similarity yield better results?

4. Which of the method/similarity-combination performs the best?

Note that it is unlikely that answers are obtained that hold for *all* datasets. See, for instance, the article of Wolpert and Macready [100]. It is likely that on different datasets, a different method will perform best. A proposed method can be considered useful if it has advantages over some (realistic) datasets. In this paper, we evaluate the methods on the Cora dataset by McCallum et al. [65], which is a commenly used benchmark. This is a dataset that contains scientific papers with their abstracts and the citation graph. From this dataset, five subsets are created. A more detailed description can be found in Section 5.2.

For every element in these subsets, a vector can be created which denotes the useful words that appear in the abstracts. This vector will be used as the annotation for the vertices. The cosine of the angle between two vertices is used as the content-based similarity measure. The value for k was varied from 100 to 2. When a clustering is found, it is compared to the clustering as defined by the labels of the vertices in the dataset. This is done by using the F-score, the harmonic mean between the precision and the recall of a cluster/label-combination. Keep in mind that k-means-NAM(A), used with the content-based similarity, is actually the regular k-means algorithm.

	CONTEXTUAL SIMILARITY		COMBINED SIMILARITY	
DATASET	ABSOLUTE DIFFERENCE	AVERAGE DIFFERENCE	ABSOLUTE DIFFERENCE	AVERAGE DIFFERENCE
CORA-1	2.4%	-1.2%	2.7%	-1.3%
CORA-2	2.4%	+1.0%	1.9%	-0.2%
CORA-3	1.6%	-0.7%	1.8%	+0.7%
CORA-4	0.9%	+0.2%	1.3%	-0.8%
CORA-5	1.3%	+0.4%	2.0%	+1.2%

Table 6.1: Percentual difference in quality of the found clusters by k-means-NAMA compared to the found clusters by k-means-NAM.

6.5 Results

6.5.1 K-means-NAM vs. K-means-NAMA

In Section 6.3.1 we hypothesized that k-means-NAMA would get similar results as k-means-NAM, but faster. This can be tested by regarding the percentual difference in score between the results with k-means-NAMA and k-means-NAM. Here, a positive percentage means that k-means-NAMA outperformed k-means-NAM, and a negative percentage means the opposite. Also the absolute value of these percentages are taken into concern. Table 6.1 shows these results.

First, the averages of all runs in a dataset for the absolute value of these percentages are small, indicating there is not a lot of difference in performance. Second, when the signs of the percentual differences are also taken into account, the differences are even smaller, indicating that one is not overall better than the other. These conclusions hold for both the contextual and the combined similarity.

Table 6.2 shows the average computation time that each method needed to perform one clustering. With the content-based similarity, there is not much difference. This is as expected, since here, k-means-NAM and k-means-NAMA both boil down to regular k-means. It could be said that k-means-NAMA is a little slower, but this is explainable by the implementation of the algorithm and it is not considered a significant difference.

When the hybrid similarities are applied, a big difference in time occurs. With respect to the content-based similarity, k-means-NAMA takes only 10 to 20 percent more time than the content-based similarity, but k-means-NAM takes about four times as much time with the contextual similarity and about five times as much time with the combined similarity. This makes k-means-NAMA roughly three times faster than k-means-NAM when the contextual similarity is applied, and roughly four times faster when the combined similarity is applied.

Since there is no real difference in quality between k-means-NAM and k-means-NAMA, from now on we only consider the results for k-means-NAMA.

6.5.2 Quality Improvement

Figure 6.2 shows the results for clustering the subsets Cora-1 and Cora-2, and Figure 6.3 shows the results for clustering the subsets Cora-3, Cora-4, and Cora-5. Table 6.3 shows the best found F-scores for all datasets for k-medoids and k-means-NAMA. Also, it shows the k for which this F-score was found. This table clearly shows that using the hybrid similarities improves the F-score significantly, as compared to using the content-based similarity. This is the case for both k-medoids and k-means-NAMA.

When k-means-NAMA is compared to k-medoids, it shows that k-means-NAMA outperforms k-medoids. Also, the k for which the found F-score is maximal, is much closer to the actual number of classes in the dataset, when using k-means-NAMA instead of k-medoids.

6.6 Conclusions

We have discussed how a hybrid similarity can be used with centroid-based clustering methods in an annotated graph. Although k-means cannot be employed in a straightforward way because the concept of a "mean node" cannot be defined, it can be approximated by two newly proposed methods. These two methods boil down to using approximate similarity measures so that k-means becomes applicable. With k-medoids, the hybrid similarities can be used without any difficulties.

The main conclusions from this work are:

1. Despite the fact that the hybrid similarities can only be applied on approximate versions of k-means, this setting still will lead to better results.

2. Of these approximate versions of k-means, there is no significant difference in quality of the found clusterings, but k-means-NAMA is much faster than k-means-NAM.

3. The hybrid similarities can be used to improve k-medoids.

4. When clustering these subsets of Cora, k-means-NAMA with the combined similarity leads to the best results. This does not mean that it always will be the best setting for every dataset. It does show, however, that the more sophisticated versions k-means-NAM(A) were worth developing, since they work better than the more straightforward approach of applying k-medoids.

Content-based Similarity					
	k-means-NAM		k-means-NAMA		Factor
Dataset	Time	Factor	Time	Factor	nama/nam
Cora-1	$2.8 \cdot 10^0$	N/A	$3.3 \cdot 10^0$	N/A	1.19
Cora-2	$1.8 \cdot 10^1$	N/A	$2.3 \cdot 10^1$	N/A	1.23
Cora-3	$3.7 \cdot 10^1$	N/A	$4.6 \cdot 10^1$	N/A	1.24
Cora-4	$1.3 \cdot 10^2$	N/A	$1.6 \cdot 10^2$	N/A	1.20
Cora-5	$4.2 \cdot 10^2$	N/A	$5.0 \cdot 10^2$	N/A	1.18

Contextual Similarity					
	k-means-NAM		k-means-NAMA		Factor
Dataset	Time	Factor	Time	Factor	nama/nam
Cora-1	$1.2 \cdot 10^1$	4.17	$4.2 \cdot 10^0$	1.28	0.37
Cora-2	$5.8 \cdot 10^1$	3.13	$2.3 \cdot 10^1$	1.02	0.40
Cora-3	$1.5 \cdot 10^2$	4.12	$5.0 \cdot 10^1$	1.08	0.33
Cora-4	$5.0 \cdot 10^2$	3.71	$1.8 \cdot 10^2$	1.12	0.36
Cora-5	$1.7 \cdot 10^3$	4.09	$6.2 \cdot 10^2$	1.23	0.36

Combined Similarity					
	k-means-NAM		k-means-NAMA		Factor
Dataset	Time	Factor	Time	Factor	nama/nam
Cora-1	$1.5 \cdot 10^1$	5.56	$4.2 \cdot 10^0$	1.28	0.28
Cora-2	$8.1 \cdot 10^1$	4.38	$2.2 \cdot 10^1$	0.98	0.28
Cora-3	$1.9 \cdot 10^2$	5.05	$5.0 \cdot 10^1$	1.08	0.27
Cora-4	$6.5 \cdot 10^2$	4.86	$1.7 \cdot 10^2$	1.05	0.26
Cora-5	$2.1 \cdot 10^3$	4.91	$5.4 \cdot 10^2$	1.08	0.26

Table 6.2: CPU time for k-means-NAM and k-means-NAMA to do one clustering (on a Intel® Quad Core™2, 2.4GHz with 4 Gb memory), for the content-based, contextual, and combined similarities. 'Dataset' denotes the used dataset; 'Time' denotes the time in seconds; 'Factor' in the third and fifth column denotes how much more time it takes to do a clustering using that similarity with regards to the content-based similarity; 'Factor nama/nam' in the sixth column denotes the factor of the time it takes k-means-NAMA to compute a clustering with regards to k-means-NAM.

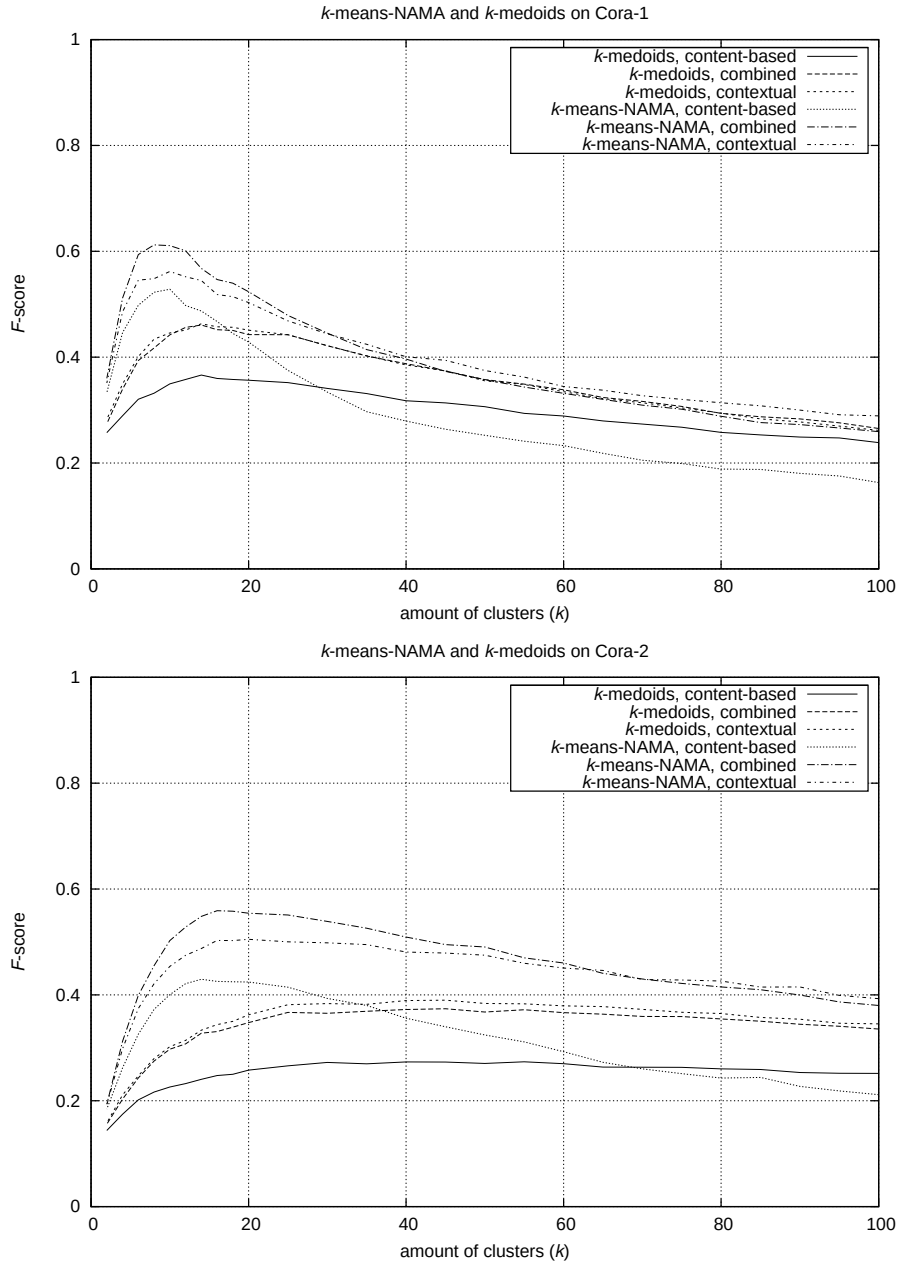


Figure 6.2: Average F-score for the resulting clusterings for different values of k, for k-medoids and k-means-NAMA on Cora-1 (top) and Cora-2 (bottom).

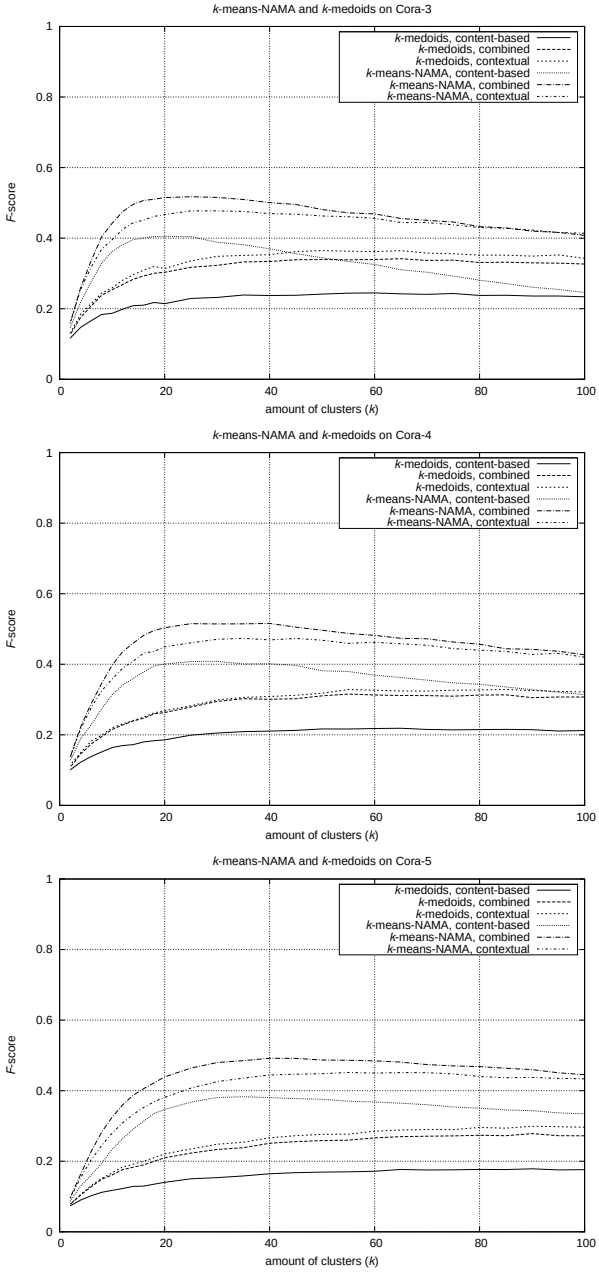


Figure 6.3: Average F-score for the resulting clusterings for different values of k, for k-medoids and k-means-NAMA on respectively CORA-3 (top), CORA-4 (middle), and CORA-5 (bottom).

k-medoids							
		Content-based		Combined		Contextual	
Dataset	$\|L\|$	F-score	k	F-score	k	F-score	k
Cora-1	8	0.37	15	0.46	13	0.46	13
Cora-2	17	0.27	40	0.37	45	0.39	45
Cora-3	24	0.24	60	0.34	60	0.36	50
Cora-4	31	0.22	65	0.32	55	0.33	55
Cora-5	45	0.18	90	0.28	90	0.30	95

k-means-NAMA							
		Content-based		Combined		Contextual	
Dataset	$\|L\|$	F-score	k	F-score	k	F-score	k
Cora-1	8	0.53	9	0.61	8	0.56	9
Cora-2	17	0.43	15	0.56	16	0.51	19
Cora-3	24	0.41	20	0.52	25	0.48	30
Cora-4	31	0.41	30	0.52	40	0.47	35
Cora-5	45	0.38	35	0.49	40	0.45	50

Table 6.3: Best F-scores found for k-medoids and k-means-NAMA, for the different similarities and datasets. Also, the appropriate k is shown. This can be compared to $|L|$ (i.e. the amount of labels in the dataset).

Chapter 7

KNN-Classification

Classification aims to determine to which class a certain element in the dataset belongs. K-nearest neighbor classification does this by regarding the k most nearest neighbors. The hybrid similarities can be used to determine which neighbors are the nearest. In this way, relational information is inserted into this method. Experiments on real life data show that using the hybrid similarities can lead to better results.

7.1 Introduction

Previous chapters showed how the hybrid similarity measure can improve standard clustering algorithms like agglomerative hierarchical clustering (Chapter 5) and k-means and k-medoids (Chapter 6). Since there are other types of data mining tasks, it is interesting to see if the hybrid similarity measure can also improve any of those. A likely candidate would be classification. In this chapter it is shown how the hybrid similarity measure can be used to improve k-nearest neighbor classification, KNN-classification.

The rest of this chapter is organized as follows. Section 7.2 explains the working of KNN-classification in detail. Section 7.3 explains how the hybrid similarity measure can be used to improve KNN-classification. Section 7.4 explains the experimental setup and the results from those experiments are reported in Section 7.5, after which Section 7.6 concludes this chapter.

7.2 The Method

There are many different classification techniques. An overview of the most important of them can be found in Section 2.2.1. Amongst them is k nearest neighbor classification, or KNN-classification. It works according to a very simple

and basic principle: To determine the label of an element, choose the one that is most prevailing amongst the k most similar elements that are labeled. It was originally proposed by Fix and Hodges [25] and in the mean time, a number of applications have been proposed, for example by Cover and Hart [18] and Han et al. [31].

More formally, considering a set of elements V, the algorithm will predict the label for an element $v \in V$ of which this label is still to be determined as follows:

1. For every element $u \in V$ with $u \neq v$ and $\lambda(u)$ is known:
2. Calculate the similarity between v and u: $\mathcal{S}(v, u)$.
3. Create the set K of size k with the property that for all $u \in K$, $w \notin K$, we have: $\mathcal{S}(v, u) > \mathcal{S}(v, w)$.
4. For every label $\ell \in \mathcal{L}$:
5. Calculate the contribution of ℓ: $C(\ell) = \sum_{u \in K} 1$ where $\lambda(u) = \ell$.
6. Now, $\lambda(v) = \ell$ where $C(\ell)$ is maximal for all $\ell \in \mathcal{L}$.

To create a good classifier, it needs to be trained on data where the labels are known. To have some sort of estimation of how good the classifier is, it needs to be tested. One problem that can occur in the training fase is overfitting. This means that the classifier is specialised in classifying items from the training set, but it does not perform too well on other examples. It is therefore important that the test dataset and the training dataset are disjoint. In this way, the amount of overfitting can be measured. This process is called cross-validation, and there are several tactics used for the process.

With k-fold cross-validation, the dataset is partioned into k random, disjoint, subsets. This k is not to be confused with the k from k nearest neighbor classification. One subset is used as the test dataset, and the others are used as training dataset. This process is repeated for every subset. The best value to choose for k remains unclear, as can be read in the book by Geisser [28]. A special case of k-fold cross-validation is when k equals the number of elements in the dataset. This is also known as leave-one-out cross-validation. Here, the entire dataset, minus one element, is used to predict the value for this one element. This is done for every element. Other examples of cross-validation include repeated random sub-sampling validation where the dataset is repeatedly and randomly split into a training set and a test set.

Cross-validation can thus be used to help determine the amount of nearest neighbors; the k in 'k nearest neighbors.' It can be seen easily that different values for k can lead to different results. This is illustrated in Figure 7.1. A low value for k makes it more susceptible to noise, since every incorrectly labeled element has a great influence on all elements it is close to. A high value for

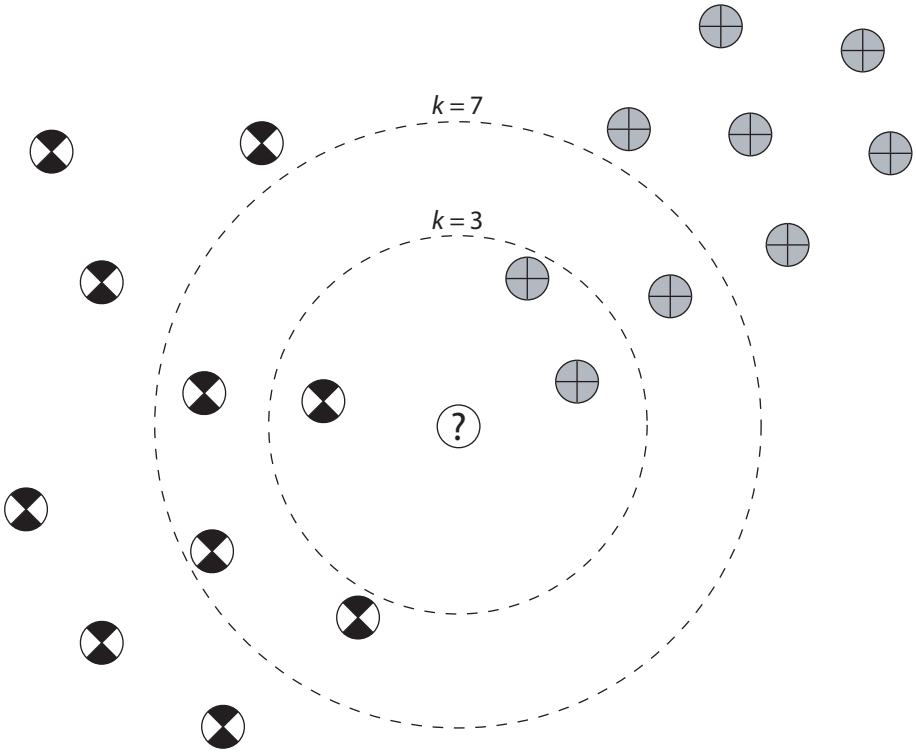


Figure 7.1: Illustration of the working of KNN-classification. The element with the question mark needs to be classified. The circles denote the areas covered for different values of k.

k makes it more difficult to draw distinctive boundaries between classes. In binary classification (where there are only two classes) it is best to choose k to be an odd number, since it will prevent ties.

When there are multiple classes, the user can choose between plurality voting or majority voting. In the first case, the class that appears most is chosen, while in the second case, a class is chosen when it appears more than half of the time amongst the k nearest neighbors. When using majority voting, there is a possibility that it is impossible to assign a class to an element. On the other hand, every class that will be assigned has a high confidence. This is not always the case with plurality voting, where it is possible that there are many different labels amongst the k nearest neighbors, and that the one that appears most still does not appear often.

It is also possible to assign a weight to the label of each neighbor according to their similarity to the element being classified. This would mean that neighbors

that are more similar have a higher influence on the label to choose. When the algorithm uses distances or dissimilarities, a monotonicly decreasing function can be used, like the inverse, one minus the distance, or the reciprocal of an exponential function.

A problem that can occur when using KNN-classification, is that classes that appear more frequently, tend to dominate the predictions, even more than their relative frequency would justify. Using weighted voting and keeping k low could prevent this from happening.

In case KNN-classification is used on a partially labeled dataset, it is possible to use the newly labeled elements to predict the label for another element. In this case it is good to keep track of the confidence of a predicted label. This can be used as a weight during the voting.

7.3 KNN-Classification and Hybrid Similarities

KNN-classification uses a similarity measure to calculate the similarity between elements. In this way, it is possible to determine which k elements are closest. Normally, the data mining algorithm would use a content-based similarity for this. When there is also relational information in the dataset, the hybrid similarities could be used to replace the content-based similarities and thus include the relational information in the algorithm.

In order to correctly use the hybrid similarities, it needs to be ascertained that they can be implemented without causing any malfunctions. As we have seen in Chapter 6, implementing the hybrid similarities in k-means was not straightforward. The similarity measure of this algorithm was not limited to similarities between elements only. It also uses the similarities between elements and some prototype that is not in the dataset. If KNN-classification only uses similarities between elements, and no other similarities, the hybrid similarities can be implemented without any further adaptions to the original algorithm.

This is indeed the case. The algorithm has two main steps: first, determine which are the k nearest neighbors, and second, determine which is the label that needs to be assigned to it. When determining which neigbors are the nearest, the algorithm computes the similarities between the element that needs to be labeled, and all other elements. After that, the k most similar elements are chosen. For this, only similarities between elements are needed, and so the hybrid similarities can be used instead of the content-based similarities.

When determining which class is the most common among these k elements, different tactics are possible. When the labels are counted unweighted, the similarities are not used, and thus, for this step, it does not matter whether the hybrid similarity is used or not. When the most common label is decided with votes that are weighted with regards to their similarity, once again, only similarities between elements are used. Therefore, here, it is also possible to use the hybrid similarities as weights to decide the label.

All in all, there does not seem to be a problem to implement the hybrid similarities. This would mean that it should be able to use them and in this way insert relational information in an algorithm that only used content-based information so far.

7.4 Experimental Setup

The experiments are done on subsets of the Cora data set by McCallum et al. [65], as described in Section 5.2. For these subsets, the datasets are defined following the definitions from Section 3.3, as described in Section 5.4.

Now, the contextual similarity can be calculated using (3.3), and the combined similarity can be calculated using (3.4). The results will be compared with the results from using the original, content-based similarity. In this way, it is possible to investigate whether the results of KNN-classification can be improved with the hybrid similarities.

The experiments are done with k ranging from 1 to $n-1$. Choosing a label for an element is done with plurality voting. The votes are weighted according to their relative similarity with regards to the element of which the label is to be determined. The experiments are done with the 'leave one out'-principle. In this case all labels are known and the label for one particular element is predicted by regarding the labels of the k closest elements and compared with the original value of the label of that element. Repeating this procedure for all elements will come to a percentage of correctly predicted labels for that combination of data set, similarity measure and k. These results are all compared.

7.5 Results

In Section 7.3, we hypothesized that the hybrid similarities could improve KNN-classification. In order to see if this is the case, experiments were done on five subsets of Cora using 'leave one out'-cross validation. During these experiments, k ranged from '1' to '$n-1$.' Figure 7.2 shows the ratio of correctly predicted labels for the subset CORA-1. The top picture shows all results, and the bottom picture zoomed in on the range of k from 1 to 40, to show this part of the graph in more detail. It is clear that the hybrid similarities outperform the original, content-based, similarity. Also, the contextual similarity outperforms the combined similarity for k greater than 4.

The experiments on the other subsets show similar results. Figure 7.3 shows these results for CORA-2 through CORA-5, zoomed in on k from 1 to 40. Here, it shows that the hybrid similarities outperform the original similarity. It also shows that for bigger k, the contextual similarity outperforms the combined similarity. Table 7.1 shows the best score for any similarity and any dataset, and for which k this score was acquired. Here, it shows that the best scores

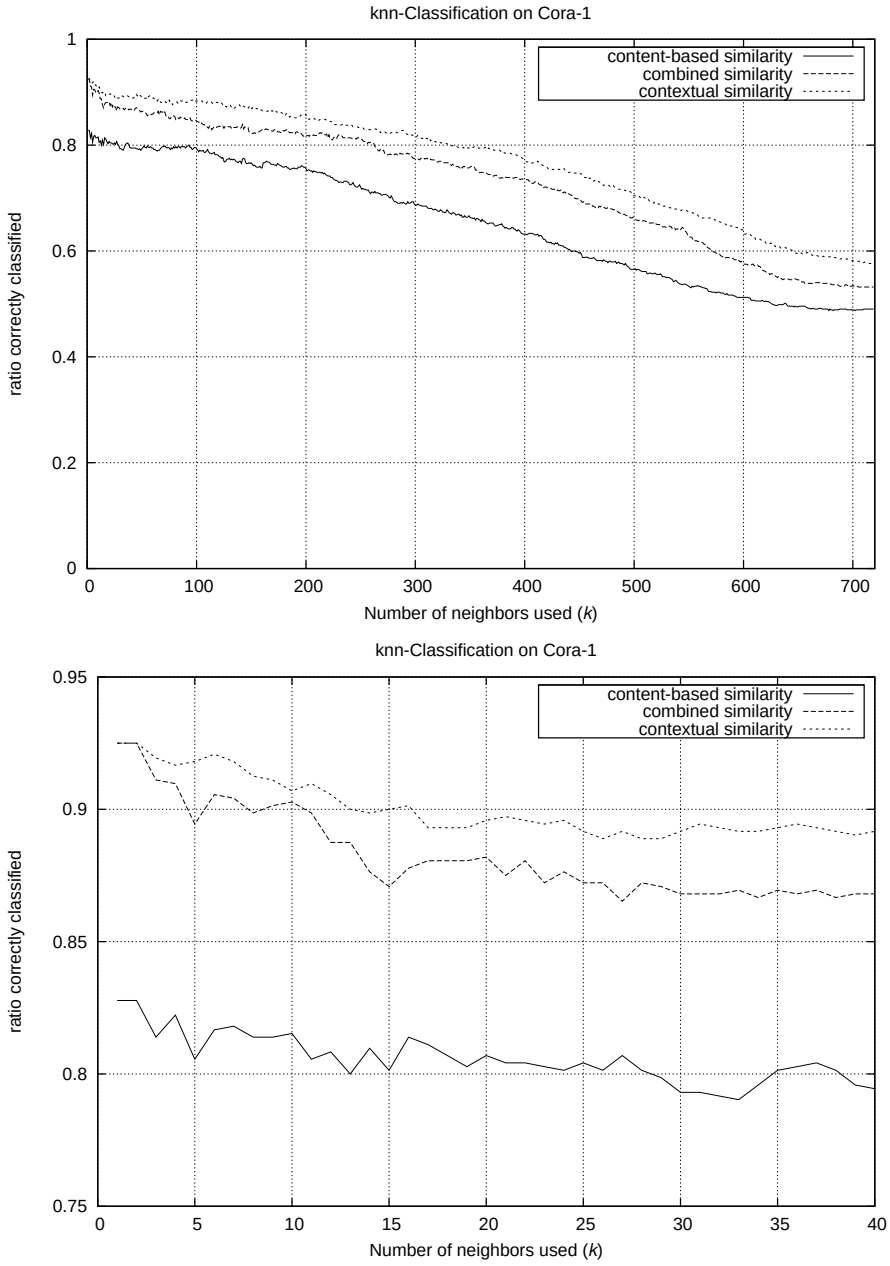


Figure 7.2: Ratio of correctly predicted labels for Cora-1. The upper picture shows the results for k is 1 to $n-1$ (where $n = 720$), and the bottom picture is zoomed in on k from 1 to 40.

Dataset	Content-based		Contextual		Combined	
	Ratio	k	Ratio	k	Ratio	k
Cora-1	0.828	2	0.925	2	0.925	2
Cora-2	0.747	10	0.881	3	0.882	3
Cora-3	0.680	5	0.823	11	0.822	3
Cora-4	0.685	12	0.815	7	0.817	4
Cora-5	0.655	14	0.773	7	0.779	5

Table 7.1: Best found scores for the three similarities on the five subsets of Cora. Also the k for which this score was acquired is given.

found for the two hybrid similarities do not differ much, but the improvement of performance with regards to the content-based similarity is significant. It is more difficult to draw any valid conclusions about the value of k for which the similarities perform best. It could be argued that the content-based similarity reaches its best performance at higher values for k than the higher similarities. Nonetheless, there are only fifteen cases and with several exceptions amongst them, this conclusion might be a bit too farfetched.

7.6 Conclusion

When a data mining tool is used on a dataset that has both content-based and relational information, it is preferred that this data mining tool could explore the entire data space. While many data mining tools are designed for only one such type of data, the hybrid similarities could be used to insert relational information into existing data mining tools that were designed only for content-based information. It has been shown that for various clustering algorithms, these hybrid similarities indeed lead to better results.

Despite its simplicity, KNN-classification is used widely as a classification technique. The core of the algorithm is the use of a similarity measure to determine which k elements are most similar to the one that is to be classified. Normally, it uses a content-based similarity, but instead of that, the hybrid similarity also can be used. In this chapter we have presented experiments that showed that using the hybrid similarities on KNN-classification can lead to better results than using a similarity measure that is based on the content of the elements alone.

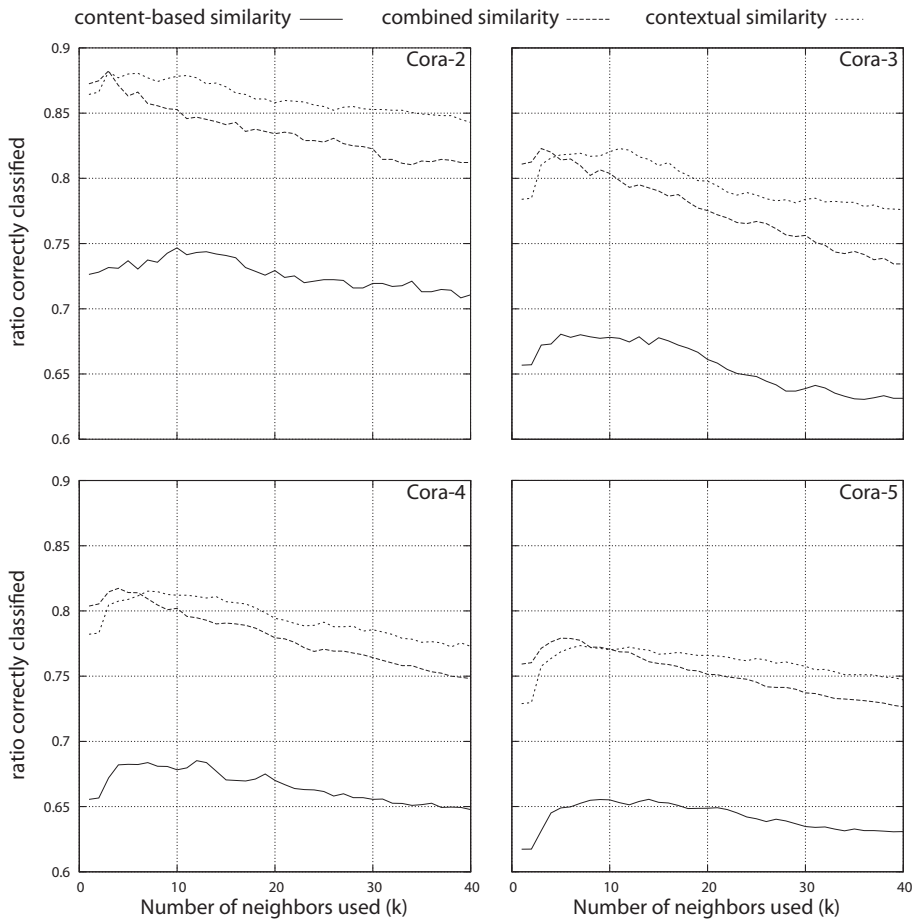


Figure 7.3: Ratio of correctly predicted labels for Cora-2, Cora-3, Cora-4, and Cora-5. They are all zoomed in on k from 1 to 40.

Part III

Analysis

Chapter 8

Reducing the Impact of Content Variability

The hybrid similarities give a better approximation of the semantic similarity between two elements. Theoretical analysis showed that the hybrid similarities use homophily to compensate for the content variability in the dataset. Experiments on a synthetic dataset, where these two concepts can be regulated, can provide an experimental validation for this. Experimental research will show that the hybrid similarities are less susceptible to content-based noise.

8.1 Introduction

Previous chapters showed that the hybrid similarities can be used to improve existing data mining techniques. Chapter 4 gave a theoretical motivation for why it could be that the hybrid similarities can get better results than the content-based similarity. It stated that, in a dataset where there is much homophily, the neighborhood of an element can also be a good estimator for that element. This is especially the case when there is much content variability in the dataset, so the content of a single element could deviate a lot from the expected form.

In this chapter we present an experimental approach to investigate the influence of the amount of homophily and content variability in a dataset on the performance of the hybrid similarities with respect to the results of a content-based similarity. The amount of homophily can be adjusted by changing relational information, and the amount of content variability can be adjusted by changing content-based information. When this is done in real-life data, it is difficult to predict whether a particular change in the content/relations will lead to an increase or a decrease of the content variability/homophily. There-

fore, in order to regulate the amount of homophily and content variability in a controlled fashion, a synthetic dataset was created.

The rest of this chapter is organized as follows. Section 8.2 explains the problem that we are trying to investigate in more detail. Section 8.3 gives the experimental setup, where Section 8.3.1 describes the used synthetic dataset and Section 8.3.2 briefly describes the used clustering methods. Section 8.4 shows the results from the experiments and to finalize, Section 8.5 draws the conclusions from these results.

8.2 Problem Setting

It can be difficult for a computer program to get the real, semantic meaning of an element precisely. There can be many reasons for that, which are described in Section 4.4. When the computer has the wrong 'understanding' of the semantics of a particular element, it will most probably calculate the similarities between that element and others incorrectly. As a result, data mining techniques that use these similarities will make wrong decisions regarding this element.

Since there is a difference between observed similarity and semantic similarity, and since we are really interested in semantic similarity, the question arises: can we find a better approximation to the semantic similarity than this observed similarity? This would of course be possible by changing the annotation space $\mathcal{A}$, making it richer. However, we assume that $\mathcal{A}$ and the data elements in it are given and cannot be changed. (The question of how to change $\mathcal{A}$ is essentially the feature construction problem, which is orthogonal to the method proposed in this thesis.) Fortunately, when the elements are also linked together, we may be able to obtain a better approximation of the semantic similarity by taking the link structure into account.

It could be that the annotation of a particular element is a bad approximation of the actual, real-life content of that element, and thus any measured similarity between that element and any other element is a bad approximation of the semantic similarity. In many natural datasets, linked elements tend to be more similar, and similar elements tend to be linked. This principle is called homophily and is described more elaborately in Section 4.2. This means that the neighborhood of an element usually consists of elements that are similar, and thus from the same class. So, in the case of an element where its annotation is a bad approximation of its actual content, the annotations of its neighbors might be better approximations than its own annotation. Therefore, in this case, it could be good to also take the similarities to the neighbors of this element into account.

The hybrid similarities can create a better approximation of the semantic similarity. This has been described more elaborately in Chapter 4. However, in this chapter, we aim to give an experimental validation for this. We try to do

so by measuring what the influence is of these bad annotations on the hybrid similarities, compared to their influence on a content-based similarity.

8.3 Experimental Setup

8.3.1 The Synthetic Dataset

In order to test and characterize the relative performance of $S_{content}$, $S_{context}$, and $S_{combined}$, the similarity measures described in Section 3.4, and to determine under which circumstances which measure works best, we have created a synthetic data set in which we can vary the amount of content variability and homophily in the network. Roughly, one could think of the dataset as a set of documents; each document is represented as a Boolean vector that shows which words occur in the document, and documents may be linked to each other. The documents are organized into classes.

We start with a dataset with zero content variability and perfect homophily. Zero content variability means that each document of a particular class is the same (has all the words characteristic for the class, and no words characteristic for other classes). Perfect homophily means that within a class all documents are linked to each other, and there are no links between classes. Afterwards, we increase content variability by removing words from documents and introducing words from other classes into them. Similarly, we decrease homophily by introducing links between documents of different classes and removing links between documents from the same class. Our goal is to investigate the effect of these changes on the accuracy of the similarity measures.

More specifically, our dataset $D = (A, G)$ consists of 1000 elements. It is described by two matrices: A, which denotes the content-based information, and G which denotes the relational information. The content of an element is described by a binary vector with 1000 components, each of which represents one particular word; each class has 100 words that are characteristic for it, and each word is characteristic for only one class. All these vectors are combined in A which is a 1000×1000 matrix where each element a_{ij} is a Boolean that states whether the j-th word appears in the content of i-th element. The relational information is described in the 1000×1000 adjacency matrix G where each element g_{ij} is a Boolean that states whether the i-th and the j-th element are connected.

The elements are divided into 10 classes with 100 elements each. The elements are arranged in an orderly manner, so the class of an element v_i is defined by $class(v_i) = \lfloor \frac{i-1}{100} \rfloor$. The same holds for the words that are characteristic to a class, so every word a_i is characterizing class for a word is defined by $class(a_i) = \frac{i-1}{100} \rfloor$. In the original dataset, there is perfect homophily, and no content variability, so A and G are both block matrices. This means that the elements a_{ij} from A are 1 when $class(v_i) = class(a_j)$, and 0 otherwise.

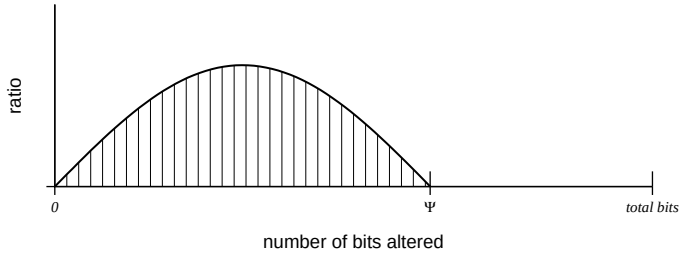


Figure 8.1: Distribution of the amount of bits that will be flipped.

Similarly, the elements g_{ij} from G are 1 when $class(v_i) = class(v_j)$, and 0 otherwise.

Now, increasing content variability means flipping bits in A, and decreasing homophily means flipping bits in G. The amount of bits that is flipped is determined by a parameter p (where relevant, we write p_A and p_G for the corresponding parameter for A and G), with $0 \leq p \leq 1$. Originally, we flipped bits randomly, giving each single bit a probability p of being flipped. Unfortunately, this led to artifacts in the results. These artifacts were probably caused by the fact that each document has approximately the same amount of flipped bits (a narrow binomial distribution around 1000p). Therefore, instead, we have chosen p to be the maximum fraction of bits flipped, but to let the actual number of flipped bits vary between 0 and $\Psi = 1000p$, according to a sinus-shaped distribution:

$$f(x) = \begin{cases} \frac{\pi}{2\Psi} \sin\left(\frac{\pi x}{\Psi}\right) & \text{if } x \in [0, \Psi] \\ 0 & \text{otherwise} \end{cases} \tag{8.1}$$

This distribution (visualized in Figure 8.1) was chosen ad hoc; the actual distribution does not matter much, the important thing is to have an easily computable distribution that is not too narrow and gradually approaches zero near the borders.

We will alter p_A and p_G separately. We expect that increasing p_A (content variability) renders the content-based similarity less accurate, while the other similarities suffer less from this. Conversely, increasing p_G is expected to have no influence on content based similarity, but cause the other similarities to deteriorate.

8.3.2 Clustering Methods

We can evaluate the practical usefulness of the similarity measures by using them to perform clustering; a similarity measure is considered more relevant if it yields clusters that are closer to the actual (hidden) classes. We have

used three different clustering algorithms: agglomerative hierarchical clustering using average linkage (hereafter referred to as agglomerative clustering), an approximate version of k-means (k-means-NAMA (See Chapter 6), hereafter referred to as k-means), and k-medoids.

We express the quality of a clustering using the *F-measure*, as suggested by Larsen and Aone [56]. This is the harmonic mean between the precision and the recall of a cluster/label-combination. It is described in detail in Section 5.4. Since there are 10 classes in the data set, we choose $k = 10$ for k-means and k-medoids, while the hierarchical clustering is simply cut off at 10 clusters.

8.4 Results

8.4.1 Results on the Synthetic Dataset

We ran experiments for many combinations of p_A and p_G. In the Figures 8.2 through 8.4 the overall results are presented. For any particular combination of p_A and p_G, they also show which similarity had the highest F-score. Of course, changing the value for p_G should have no effect on the performance of the content-based similarity since it does not use the relational information. Any small differences in results are due to random choices that the algorithm needs to make. In the case of k-means and k-medoids, this is the choice for the initial k prototypes. In the case of agglomerative hierarchical clustering, this is the choice which two clusters to merge when there is more than one pair of elements that is the most similar.

For agglomerative hierarchical clustering, Figure 8.2 shows that there is not much difference between the results of the content-based similarity and the combined similarity, but the contextual similarity behaves very differently. When there is little content-based noise ($p_A \leq 0.3$), the content-based and combined similarity reach the perfect score, but the contextual similarity has difficulties as the relational noise increases. On the other hand, when there is only a small amount of relational noise, the contextual similarity starts to outperform the two others when p_A increases.

Figure 8.3 shows the results for k-means. To some extent, they are comparible to the results for agglomerative hierarchical clustering, but the resulting behaviour of the contextual similarity is even more profound here. Once again, the content-based similarity and the combined similarity do not differ much. They are both hardly influenced by the relational noise. Also, they perform very well for low values of p_A, and start to drop when p_A increases. The difference here is that the results of the combined similarity do not drop so fast as those of the contextual similarity (especially when p_G is low).

The contextual similarity behaves very differently. It is hardly influenced by noise on the content, except when the relational noise is high. Furthermore, it performs very well for low values of the relational noise and starts decreasing

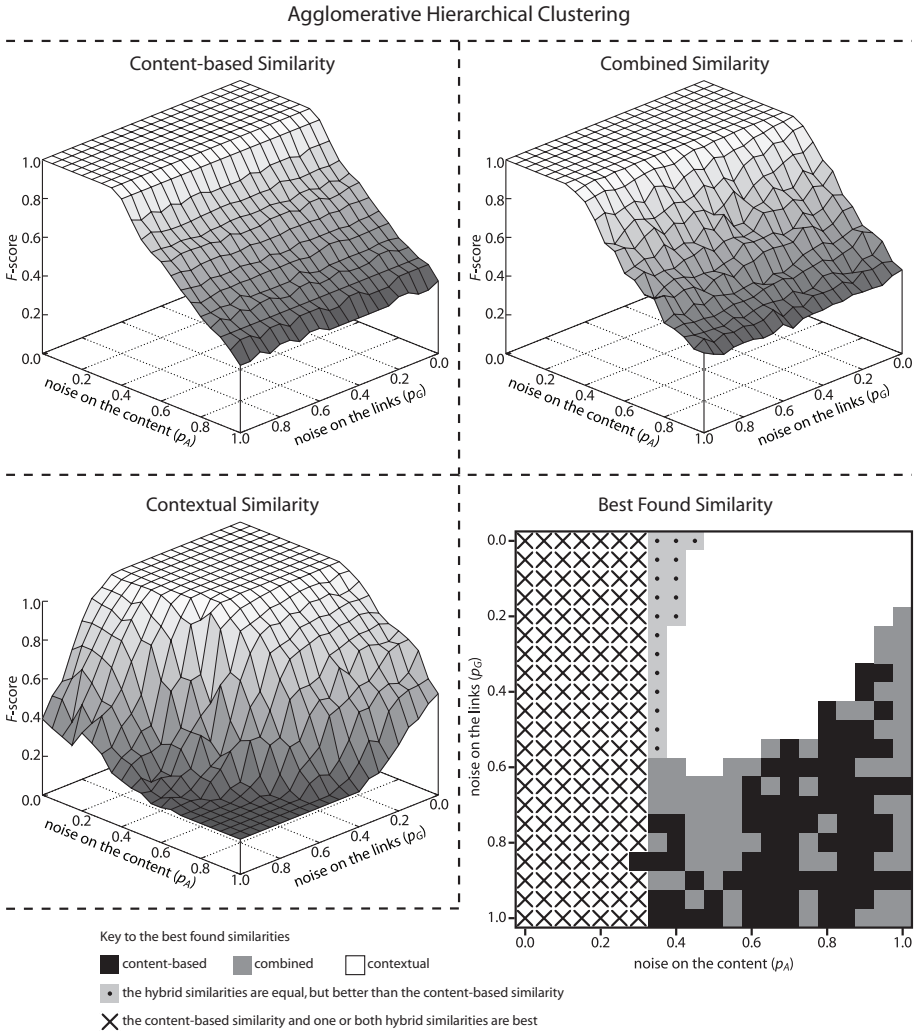


Figure 8.2: Results for agglomerative hierarchical clustering. The three-dimensional graphs give the found F-scores for that combination of p_A and p_G. The two-dimensional graph tells which similarity had the highest F-score for each combination of p_A and p_G. Special symbols are used when more than one similarity is the best. The vertical axis in the bottom-right panel is reversed to keep the same orientation as the three-dimensional graphs.

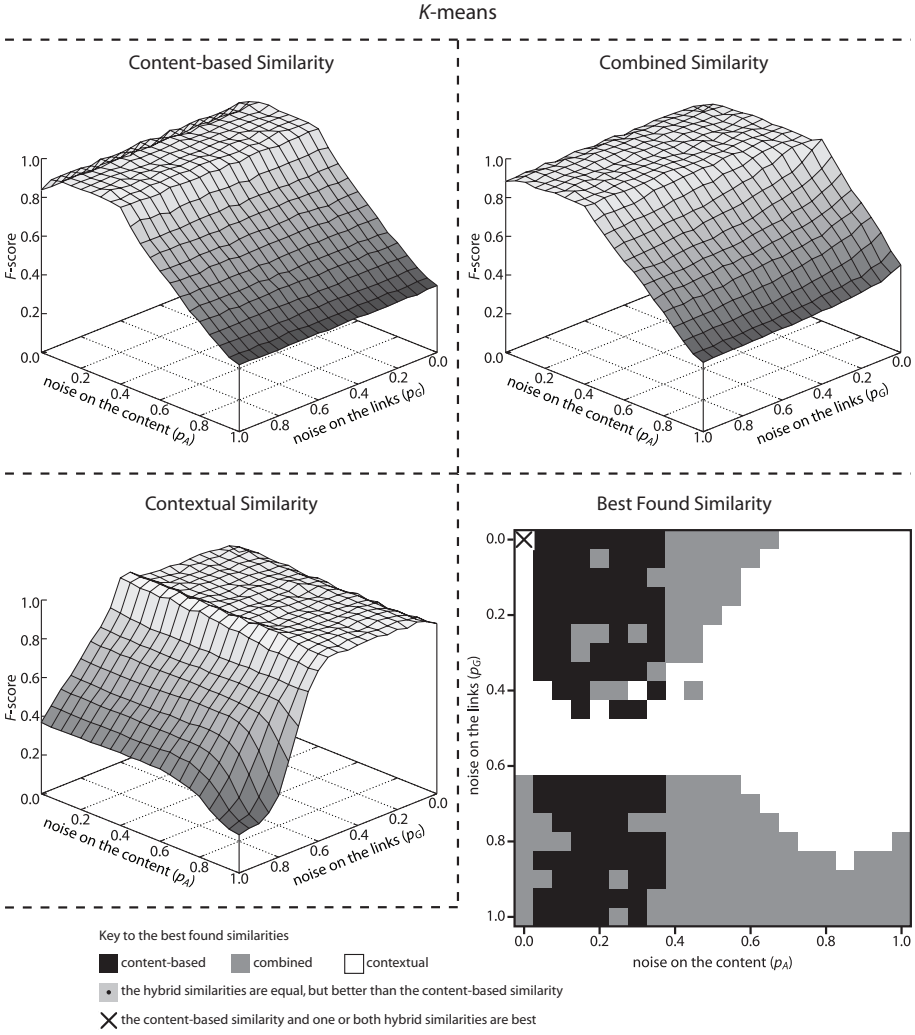


Figure 8.3: Results for k-means. The three-dimensional graphs give the found F-scores for that combination of p_A and p_G. The two-dimensional graph tells which similarity had the highest F-score for each combination of p_A and p_G. Special symbols are used when more than one similarity is the best. The vertical axis in the bottom-right panel is reversed to keep the same orientation as the three-dimensional graphs.

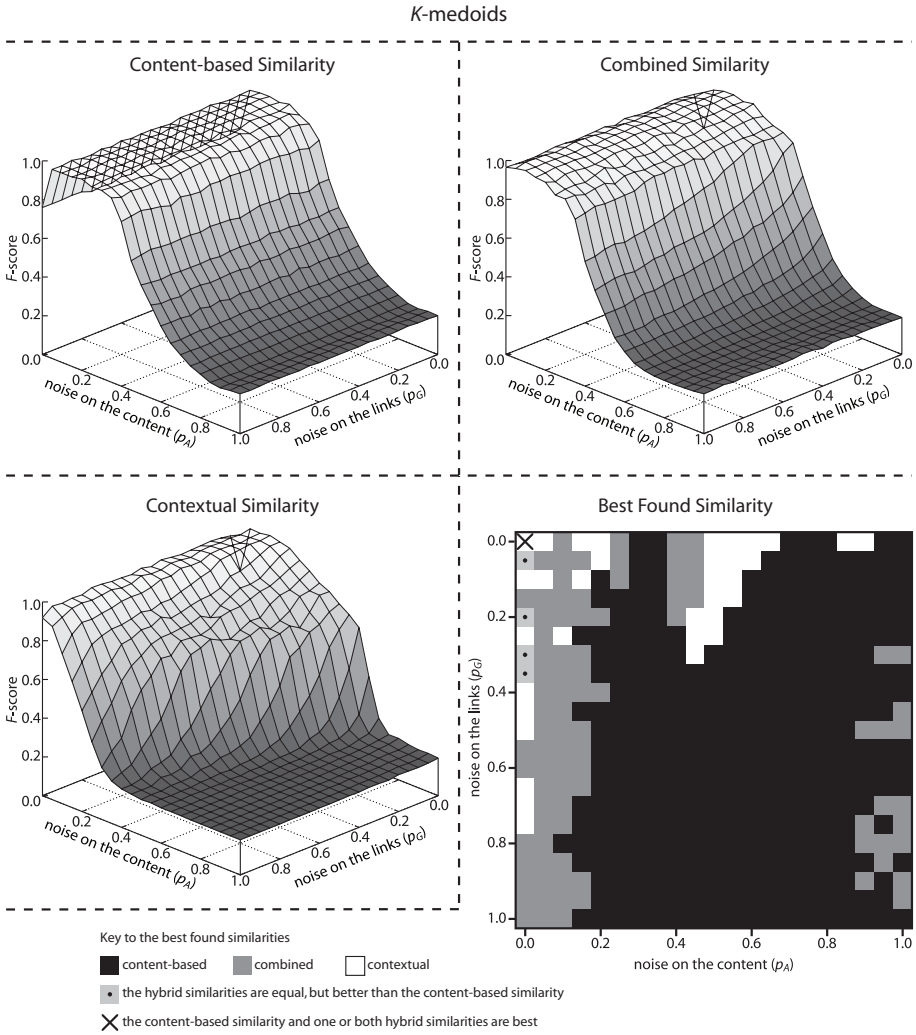


Figure 8.4: Results for k-medoids. The three-dimensional graphs give the found F-scores for that combination of p_A and p_G. The two-dimensional graph tells which similarity had the highest F-score for each combination of p_A and p_G. Special symbols are used when more than one similarity is the best. The vertical axis in the bottom-right panel is reversed to keep the same orientation as the three-dimensional graphs.

when this noise is high ($p_G > 0.6$). There is a peak in performance in a small range ($0.5 \leq p_G \leq 0.6$) where the contextual similarity outperforms the other similarities, even for low values of p_A. It is uncertain what causes this peak.

Figure 8.4 shows the results for k-medoids. These figures show similar characteristics. Again, the content-based similarity and the combined similarity do not differ much from each other; they both hardly are influenced by the amount of relational noise, both perform very well with low content-based noise, and both their performances start to drop when p_A rises. The difference this time is that here, the content-based similarity slightly outperforms the combined similarity most of the time.

On the other hand, the behaviour of the contextual similarity is very different from what we have seen before. It performs poorly, and only in a few cases does it manage to get the best F-score. This is in the area where relational noise is low, and there is a fair amount of content-based noise.

8.4.2 Results on Real Data

The above analysis used synthetic data, where homophily and content variability were controlled. It is clear that, depending on the amount of homophily and content variability, one method will be better than the other. The question remains in which areas certain real-life datasets lie. Earlier work with the Cora dataset (See Chapters 5 through 7) has shown that the area where contextual and combined similarity perform better is certainly not empty. This, in turn, implies that some networks indeed contain enough homophily to make the use of hybrid similarities rewarding.

8.5 Conclusion

When estimating the similarity between two annotated nodes in a graph, one can simply rely on the nodes' annotations. However, we have argued that it may be useful to take also the network context into account. Using a synthetic dataset, we have observed experimentally what the effects of homophily and content variability on the different similarities are. The precise effects differ from one data mining technique to another.

Several general conclusions can be drawn. First, the content-based similarity and the combined similarity do not differ much in behaviour. Second, the contextual similarity outperforms the others when the amount of relational noise is small, and the amount of content-based noise is high. This shows that this similarity measure indeed behaves as expected by our theoretical analysis. This confirms and explains why similarity measures that look not only at content but also at graph context are of practical importance.

The question that rises is: to what extent does the synthetic dataset behave as a real-life dataset? The experiments on subsets of the Cora dataset, as de-

scribed in previous chapters, show some important differences. First of all, on Cora, the two hybrid similarities behaved more or less the same. This is opposed to the synthetic dataset, where it is the content-based and the combined similarity that behave similarly. Secondly, during the experiments on the subsets of Cora, the hybrid similarities outperformed the content-based similarity. However, with the synthetic dataset, this was not the case.

This could mean that the Cora dataset probably has the ratio of homophily and content variability where the hybrid similarities outperform the content-based similarity. The region where this is consistent for the three data mining techniques is where there is much homophily (p_G is low) and much content variability (p_A is high). In the Cora dataset, the content is defined by the words that are in the abstracts. Since abstracts in general are relatively small pieces of text, it is likely that there is indeed much content variability in the Cora dataset. The relational information comes from the citations. Usually, most citations from or to a paper are of papers that are about the same subject. Therefore it is also likely that there is much homophily in the Cora dataset. All in all, it seems that Cora is in the right range for the hybrid similarities to perform well.

Chapter 9

Different Versions of Hybrid Similarity Measures

The contextual similarity is defined by first taking the average similarity to the neighbors on either side, and then taking the average of these two. The combined similarity is defined as the average of the contextual and the content-based similarity. Nonetheless, there are other ways to combine all these similarities. Theoretical analysis discovers five different conceptual options to do so. Experimental research shows that there is no significant difference in results between any of them.

9.1 Introduction

When taking a closer look at the way in which the contextual similarity between elements v, w from a set of elements V is calculated (as described in Section 3.4) it can be seen that first one takes the average of all similarities from v to the neighbors of w, known as $\mathcal{S}_{neighbor}(v, w)$ from (3.2). Then, one does the same for all similarities from w to the neighbors of v, thus calculating $\mathcal{S}_{neighbor}(w, v)$. Finally, as described in (3.3), the average of these two values is taken to get the contextual similarity.

This means that $\mathcal{S}_{neighbor}(v, w)$ and $\mathcal{S}_{neighbor}(w, v)$ are contributing equally to $\mathcal{S}_{context}(v, w)$. This sounds like a fair deal. Since there is no demonstrable difference between v and w, why should $\mathcal{S}_{neighbor}(v, w)$ and $\mathcal{S}_{neighbor}(w, v)$ be treated differently? But what if the amount of neighbors that v has is much higher than the amount of neighbors that w has? In that case, the similarity between v and one of the few neighbors of w has a much greater contribution to the contextual similarity than the similarity between w and one of the many neighbors of v.

It can easily be seen that this situation may not be so desirable. As stated in Chapter 8, one of the benefits of the hybrid similarity measures is their ability to compensate for noisy information by taking the average of several similarities in the neighborhood defined by the relational information. The neighborhood similarity from one element to an element with a lot of neighbors is therefore much more reliable than the neighborhood similarity from that element to an element with only a few neighbors. This difference in reliability does not show in the calculation of the contextual similarity, since it regards the neighborhood similarity from one element to the other the same as the neighborhood similarity the other way around, regardless their reliability.

With this in mind, it may not be so fair at all that the contributions of $\mathcal{S}_{neighbor}(v, w)$ and $\mathcal{S}_{neighbor}(w, v)$ are equal. Much could be said for a hybrid similarity measure that is calculated in such a way that the contribution of a more reliable similarity is higher than that of a less reliable similarity. This can be solved easily by calculating a hybrid similarity between two elements as the average of all similarities from either element to the neighbors of the other. Then every similarity from one element to a neighbor of the other contributes equally, regardless of the amount of neighbors that element has.

This shows that there are more possibilities to create a hybrid similarity from the similarities between one element and the neighbors of the element between which the hybrid similarity is needed. It also shows that these new hybrids can have an interesting meaning, despite the fact that it is different from the original ideas. This raises the question of what different hybrid similarities can be created in comparable ways and which performs best.

In this chapter, we take a closer look at what the impact of those different choices is. Where this results in different similarities, we test if these similarities will lead to different results when applied in clustering or classification algorithms. From this, we will try to find the best way to calculate a similarity between two elements, based on their content and neighborhoods (if such best way exists).

The rest of this chapter is organized as follows. Section 9.2 analyses the contextual and combined similarities and from there, derives five meaningful and conceptually distinct manners to combine all used similarities into a hybrid similarity. Section 9.3 describes the experimental setup. Section 9.4 gives the results from these experiments, and Section 9.5 draws conclusions from these results.

9.2 More Similarities

First, the original hybrid similarities are analysed so that it will be easy to derive a variety of new hybrid similarity measures from it. Once again, consider the data set D defined as $D = (V, E, \alpha, \lambda)$ where $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vertices or elements, $E \subseteq V \times V$ is the set of edges, $\alpha : V \to \mathcal{A}$ a function

that assigns to any element v of V an "annotation", and $\lambda : V \to \mathcal{L}$ a function that assigns to any element v of V a label. The annotation $\alpha(v)$ is considered to be the *content* of vertex v and the label $\lambda(v)$ is considered to be its *class*.

This data set format is the same as the one described in Section 3.3. Again, the space of possible annotations is left open and can be anything as long as it is possible to define a similarity measure $\mathcal{S}_{content} : \mathcal{A} \times \mathcal{A} \to \mathbb{R}$ as a function that assigns a value to any pair of annotations expressing the similarity between these annotations. The space of possible labels is a set of m distinct elements, thus, $\mathcal{L} = \{l_1, l_2, \ldots, l_m\}$, each representing a specific class. It is possible that the class of a certain element is unknown.

Now, let $\mathcal{N}(w)$ be the set of all neighbors of w, so $u \in \mathcal{N}(w) \Leftrightarrow (w, u) \in E$, and let $\mathcal{T}(v, w)$ be the sum (or total) of all similarities from v to all neighbors of w, so $\mathcal{T}(v, w) = \sum_{u \in \mathcal{N}(w)} \mathcal{S}_{content}(v, u)$. To make things more clear, all definitions are also explained in Figure 9.1.

Using $\mathcal{N}$ and $\mathcal{T}$, the contextual similarity from (3.3) can be rewritten as:

$$\mathcal{S}_{contextual}(v, w) = \frac{\dfrac{\mathcal{T}(v, w)}{|\mathcal{N}(w)|} + \dfrac{\mathcal{T}(w, v)}{|\mathcal{N}(v)|}}{2} \tag{9.1}$$

Using this contextual similarity, the combined similarity from (3.4), with $c = \frac{1}{2}$, can be rewritten as:

$$\mathcal{S}_{combined}(v, w) = \frac{1}{2} \cdot \frac{\dfrac{\mathcal{T}(v, w)}{|\mathcal{N}(w)|} + \dfrac{\mathcal{T}(w, v)}{|\mathcal{N}(v)|}}{2} + \frac{1}{2} \cdot \mathcal{S}_{content}(v, w) \tag{9.2}$$

From here it is easy to see that:

$$\begin{aligned}
\mathcal{S}_{combined}(v, w) &= \frac{\dfrac{\mathcal{T}(v, w)}{|\mathcal{N}(w)|} + \dfrac{\mathcal{T}(w, v)}{|\mathcal{N}(v)|}}{4} + \frac{2 \cdot \mathcal{S}_{content}(v, w)}{4} \\
&= \frac{\dfrac{\mathcal{T}(v, w)}{|\mathcal{N}(w)|} + \mathcal{S}_{content}(v, w) + \dfrac{\mathcal{T}(w, v)}{|\mathcal{N}(v)|} + \mathcal{S}_{content}(w, v)}{4} \\
&= \frac{\dfrac{\dfrac{\mathcal{T}(v, w)}{|\mathcal{N}(w)|} + \mathcal{S}_{content}(v, w)}{2} + \dfrac{\dfrac{\mathcal{T}(w, v)}{|\mathcal{N}(v)|} + \mathcal{S}_{content}(w, v)}{2}}{2}
\end{aligned} \tag{9.3}$$

This shows that three different types of averages are calculated: first, the average of the similarities from one element to all neighbors of the other element is calculated. From now on, this will be known as the *neighborhood average*. Second, the average of the neighborhood average and the content-based similarity is calculated. When Figure 9.1 is divided into a right and a left half, it is

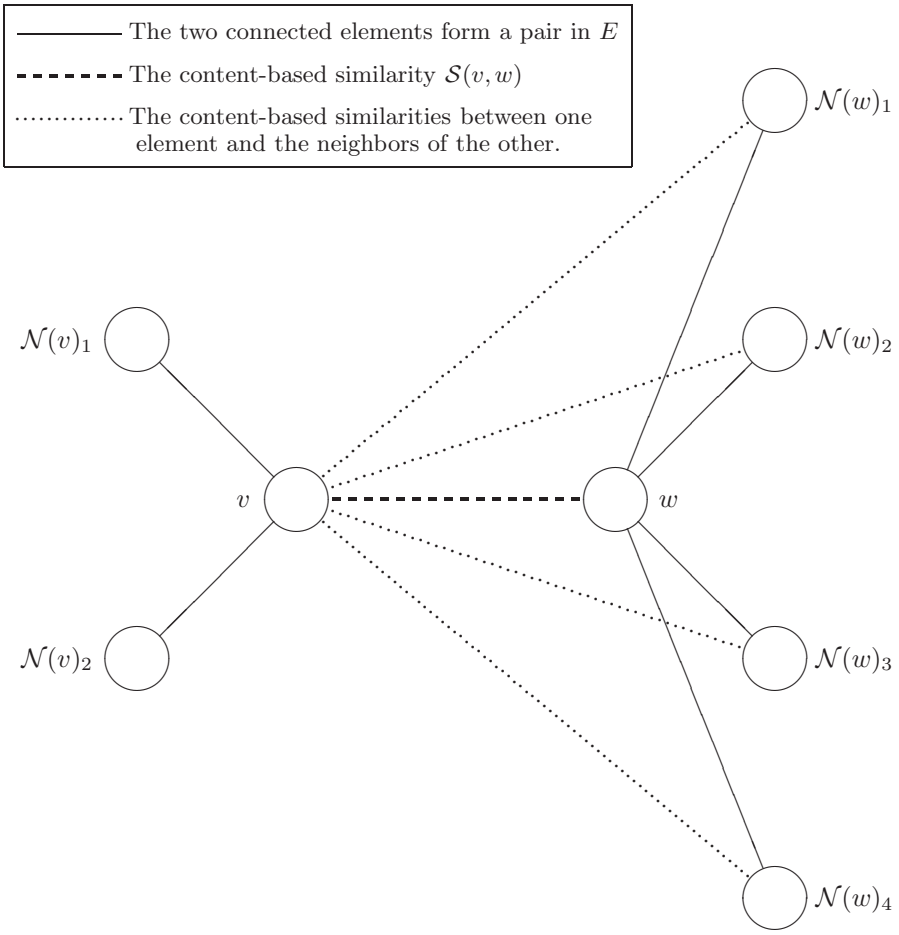


Figure 9.1: Overview of the objects necessary to calculate all different hybrid similarities between v and w. The solid lines represent the edges in the graph (i.e. the relational information from the dataset). Thus, an element labeled with $\mathcal{N}(x)_y$ is the y^{th} neighbor of x, with x being either v or w. The dashed line represents the content-based similarity between v and w ($\mathcal{S}_{content}(v, w)$). The dotted lines represent the content-based similarities between v and the neighbors of w ($\mathcal{N}(w)$). The sum of the similarities represented by dotted lines is $\mathcal{T}(v, w)$. Nota bene, the content-based similarities from w to all neighbors of v, needed to calculate $\mathcal{T}(w, v)$, are not drawn in this figure to keep it more clear. Also notice that it is possible that v and w are connected or that there is an element that is a neighbor of both v and w. Since this does not change the way to calculate any of the similarities, it is not taken into account here.

clear that this average is the average of all similarities from one element to all elements on the other side. Therefore from now on this will be known as the *side average*. Third, the average between the two sides is calculated. This will result in the average of the total amount of similarities and thus will be known as the *total average*.

It has been established that there are three places where an average can be taken: neighbor average, side average and total average. Also, it could be decided that on any of those places the average is not taken. In that case, one just sums the terms and takes the average in a later stadium. The only place where such a choice is not available is the total average. If this final average is not taken, the values for the resulting similarity could be out of range compared to the content-based similarity easily. To recall, the new hybrid similarity replaces the original content-based similarity in some data mining algorithm. If the algorithm uses similarities that are, for instance, between 0 and 1, then the values of the hybrid similarity also must be between 0 and 1.

This leaves two places where it could be decided whether or not to take the average at that point. Also, there is the choice of adding the content-based similarity between v and w, or only using the similarities between one of the nodes and the neighbors of the other. This leaves $2^{2+1} = 8$ possible hybrid similarities. Table 9.1 gives an overview of the resulting similarities from these combinations.

Table 9.1 clearly shows that, from the eight possible combinations, only five distinct hybrid similarities can be created. There are two reasons for this. First, when the content-based similarity between v and w is not included, the side average is calculated as an average only over the similarities from one element to the neighbors of the other, which is the same as the neighborhood average. In this case, there is no difference between taking the neighborhood average or taking the side average.

Second, if the neighborhood average is taken, then the neighborhood similarity (as calculated with (3.2)) could be considered as one similarity. If, in this case, the side average also is taken, then this will result in the neighborhood similarity being added to the content-based similarity and divided by 2. In the next step, the similarity for this side is added to the similarity of the other side and also divided by 2. If, on the other hand, the side average is not taken, then the two neighborhood averages are added to two times the content-based similarity. The resulting sum is divided by 4. The result of this is, of course, the same as when the side similarity was taken.

There are two different aspects for which the hybrid similarities are distinct. The first is whether or not the content-based similarity between v and w is inserted. This shows in the name of the hybrid similarity where a 'C' denotes the presence of the content-based similarity.

The second difference is how the average is calculated. It already has been shown that there is no difference between taking only the neighborhood average

#	$\mathcal{S}_{content}$	NEIGHBOR AVERAGE	SIDE AVERAGE	HYBRID SIMILARITY
1	no	no	no	$\mathcal{S}_{TA}$
2	no	no	yes	same as #3
3	no	yes	no	$\mathcal{S}_{NA}$
4	no	yes	yes	same as #3
5	yes	no	no	$\mathcal{S}_{CTA}$
6	yes	no	yes	$\mathcal{S}_{CSA}$
7	yes	yes	no	$\mathcal{S}_{CNA}$
8	yes	yes	yes	same as #7

Table 9.1: Different options for the hybrid similarities.

and taking both the neighborhood average and the side average. Also, it has been shown that the total average always needs to be taken. This leaves three possibilities: the one where the neighborhood average and the total average are taken, the one where the side average and the total average are taken, and the one where only the total average is taken. These three possibilities are distinguished by the first average that is taken and thus these averages show in the second part of the name of the hybrid similarities by, respectively: ('*NA*'), ('*SA*'), and ('*TA*').

The results in five different similarities, which we now discuss in detail.

No Content-Based Similarity and Neighborhood Average

For the first hybrid similarity, the content-based similarity between v and w is left out. Both the neighborhood average and the total average are taken, which results in the original contextual similarity as described in (9.1). Known as the Similarity without content-based similarity and Neighborhood Average ($\mathcal{S}_{NA}$), it is described as follows.

$$\mathcal{S}_{NA} = \frac{\dfrac{\mathcal{T}(v,w)}{|\mathcal{N}(w)|} + \dfrac{\mathcal{T}(w,v)}{|\mathcal{N}(v)|}}{2} \tag{9.4}$$

No Content-Based Similarity and Total Average

The second hybrid similarity is also without the content-based similarity between v and w, but here only the total average is taken. Known as the Similarity without content-based similarity and Total Average ($\mathcal{S}_{TA}$), it is described as follows.

$$\mathcal{S}_{TA} = \frac{\mathcal{T}(v,w) + \mathcal{T}(w,v)}{|\mathcal{N}(w)| + |\mathcal{N}(v)|} \tag{9.5}$$

With Content-Based Similarity and Neighborhood Average

The third hybrid similarity includes the content-based similarity between v and w. It takes both the neighborhood average and the total average, and it is the original combined similarity as described in (9.2). Known as the Similarity with Content-based similarity and Neighborhood Average ($\mathcal{S}_{CNA}$), it is described as follows.

$$\mathcal{S}_{CNA} = \frac{\frac{\mathcal{T}(v,w)}{|\mathcal{N}(w)|} + \mathcal{S}_{content}(v,w) + \frac{\mathcal{T}(w,v)}{|\mathcal{N}(v)|} + \mathcal{S}_{content}(w,v)}{4} \tag{9.6}$$

With Content-Based Similarity and Side Average

The fourth hybrid similarity uses the content-based similarity between v and w and both the side average and total average are taken. Known as the Similarity with Content-based similarity and Side Average ($\mathcal{S}_{CSA}$), it is described as follows.

$$\mathcal{S}_{CSA} = \frac{\frac{\mathcal{T}(v,w) + \mathcal{S}_{content}(v,w)}{|\mathcal{N}(w)| + 1} + \frac{\mathcal{T}(w,v) + \mathcal{S}_{content}(w,v)}{|\mathcal{N}(v)| + 1}}{2} \tag{9.7}$$

With Content-Based Similarity and Total Average

The fifth and final hybrid similarity also uses the content-based similarity between v and w, and only calculates the average at the very end where the grand total of all similarities is one big sum. Known as the Similarity with Content-based similarity and Total Average ($\mathcal{S}_{CTA}$), it is described as follows.

$$\mathcal{S}_{CTA} = \frac{\mathcal{T}(v,w) + \mathcal{S}_{content}(v,w) + \mathcal{T}(w,v) + \mathcal{S}_{content}(w,v)}{|\mathcal{N}(w)| + |\mathcal{N}(v)| + 2} \tag{9.8}$$

9.3 Experimental Setup

The five hybrid similarities need to be compared to the original content-based similarity as well as amongst each other. To do so, several experiments were conducted on the five subsets, CORA-1 through CORA-5 of the Cora data set [65] as described in Section 5.2. For three different data mining algorithms, the original content-based similarity was replaced by any of the five hybrid similarities and the effect on the performance was measured. Since all three algorithms were described more elaborately in previous chapters, only a short description is given here.

The first algorithm used is *agglomerative hierarchical clustering*, as previously described in Section 5.3 and in, for instance, the book by Tan et al. [94].

In short, agglomerative hierarchical clustering is a clustering algorithm that starts with every element in its own cluster. Then, the algorithm iteratively merges the two closest clusters until only one cluster remains. During these experimants, average linkage is used.

The second algorithm used is *k-medoids* from Kaufman and Rousseeuw [45]. It is previously described in Section 6.2.1. In short, k-medoids is a clustering algorithm that starts by selecting k random elements which start as the center, or medoid, of the clusters. Then, iteratively, every other element is placed in the cluster with the mean it is most similar to. Once all elements are assigned, the medoids are recalculated as the element in a cluster that is most similar to all others. The algorithm halts when nothing changes anymore.

The clusterings found in the above mentioned clustering algorithms are evaluated by calculating the F-score by Larsen and Aone [56]. For agglomerative hierarchical clustering this F-score is calculated every 10 steps. For k-medoids, k was varied from 100 to 2 and for every k the experiments were repeated 100 times to compensate for the effects caused by randomly choosing the initial medoids.

The third algorithm used is KNN-*classification*, as described in Section 7.2 and in, for instance, the book by Tan et al. [94]. In short, KNN-classification is a classification algorithm that tries to predict a label for a certain element. It does so by regarding the labels of the k most similar elements and choosing the label that is most common as a prediction. The ratio of correctly predicted elements defines how well it classifies.

9.4 Results

Figure 9.2 shows two examples of the results as found by the clustering algorithms: k-medoids for subset CORA-1 and agglomerative hierarchical clustering for subset CORA-3. The results of other subsets for these algorithms show similar characteristics.

The graph describing the results for k-medoids on CORA-1 shows that clustering with the original content-based similarity gives significantly weaker results than using any of the hybrid similarities. Furthermore, there is no significant difference in performance between any of the hybrid similarities.

The same can be said when regarding the results for agglomerative hierarchical clustering on CORA-3. At the beginning of the clustering algorithm, the F-scores do not differ much. The content-based similarity is even slightly better than the others. But, as the algorithm proceeds, and the F-scores rise, the content-based similarity is the first that reaches its maximum. The hybrid similarities reach their maximum at a later point, where it is also higher.

Figure 9.3 shows the results for KNN-classification on subset CORA-5. The first graph shows the results for k in the range from 1 to 7918 which is the size of CORA-5. The second graph shows the same results, but then magnified.

Here, k ranges from 1 to 40 on the horizontal axis and the vertical axis is adjusted accordingly. Once again, the difference in performance between the original similarity and the five hybrids is very obvious. Also here, there is no real significant difference in performance between the five hybrids.

Table 9.2 gives an overview of all the best results found. For every clustering method/dataset-combination, it is always the original, content-based similarity that performs the worst. The five hybrid similarities are always significantly better. The one exception might be $\mathcal{S}_{CTA}$ with k-medoids on CORA-5, which is only slightly better than $\mathcal{S}_{content}$ and closer to this content-based similarity than to the other hybrids. But, as it performs similar to the other hybrids on the other settings, it can be considered an exception.

9.5 Conclusions

The contextual similarity and the combined similarity from Chapter 3 were created with the use of the similarities to the neighbors of an element. We argued that there are five conceptual possibilities to combine all these similarities to a hybrid similarity. These five hybrid similarities have been used on three different clustering techniques: agglomerative hierarchical clustering, k-medoids, and KNN-classification. Experiments on subsets of the Cora dataset showed that:

- The hybrid similarities outperform the original, content-based similarity in every setting.
- There is no significant difference between the hybrid similarities.

From this, it can be concluded that it does lead to better results when a content-based similarity is replaced by a hybrid similarity (i.e. a similarity that is a combination of the similarities from one element to the neighbors of the other). However, it does not matter in which way these similarities are combined.

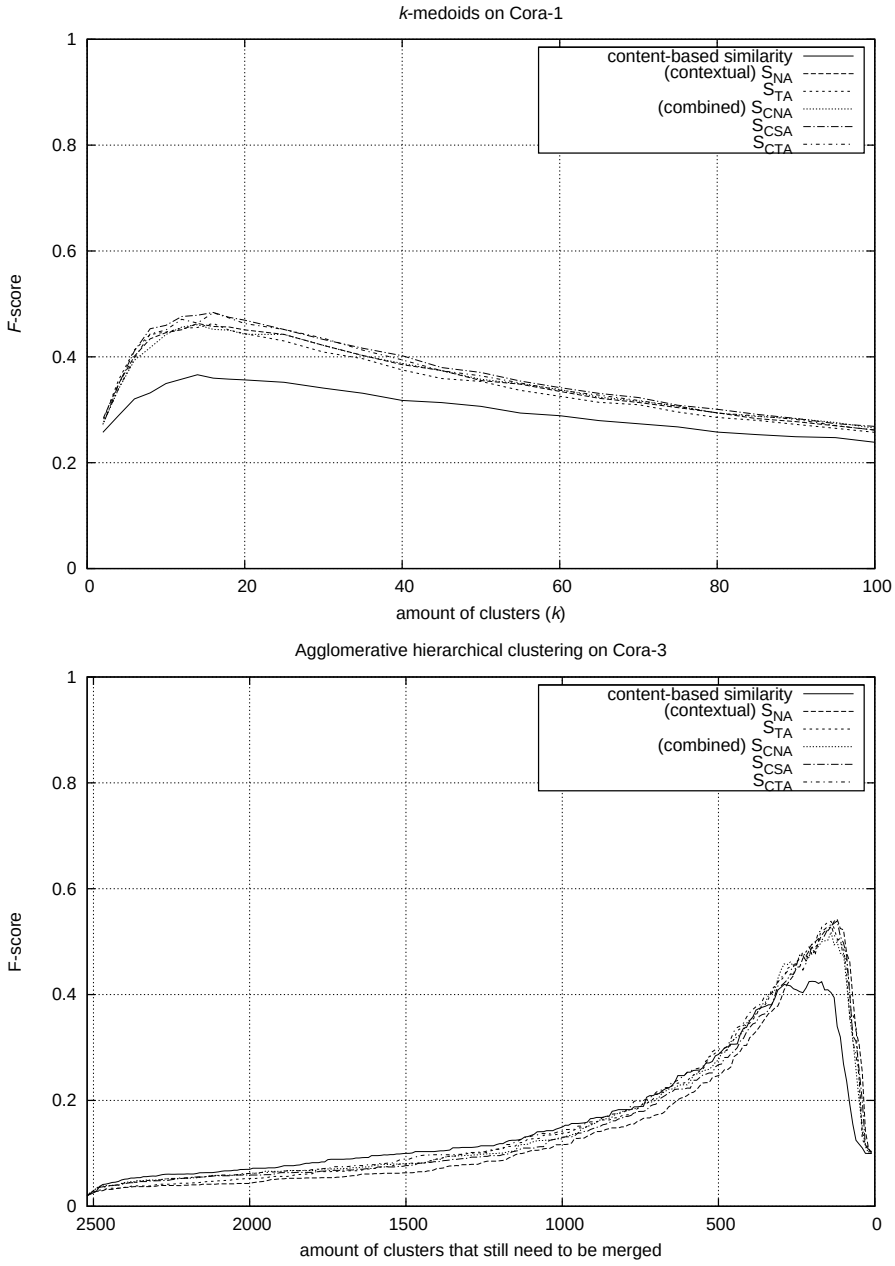


Figure 9.2: Clustering quality for k-medoids on CORA-1 (top) and agglomerative hierarchical clustering on CORA-3 (bottom) for the five hybrid similarities compared to the content-based similarity.

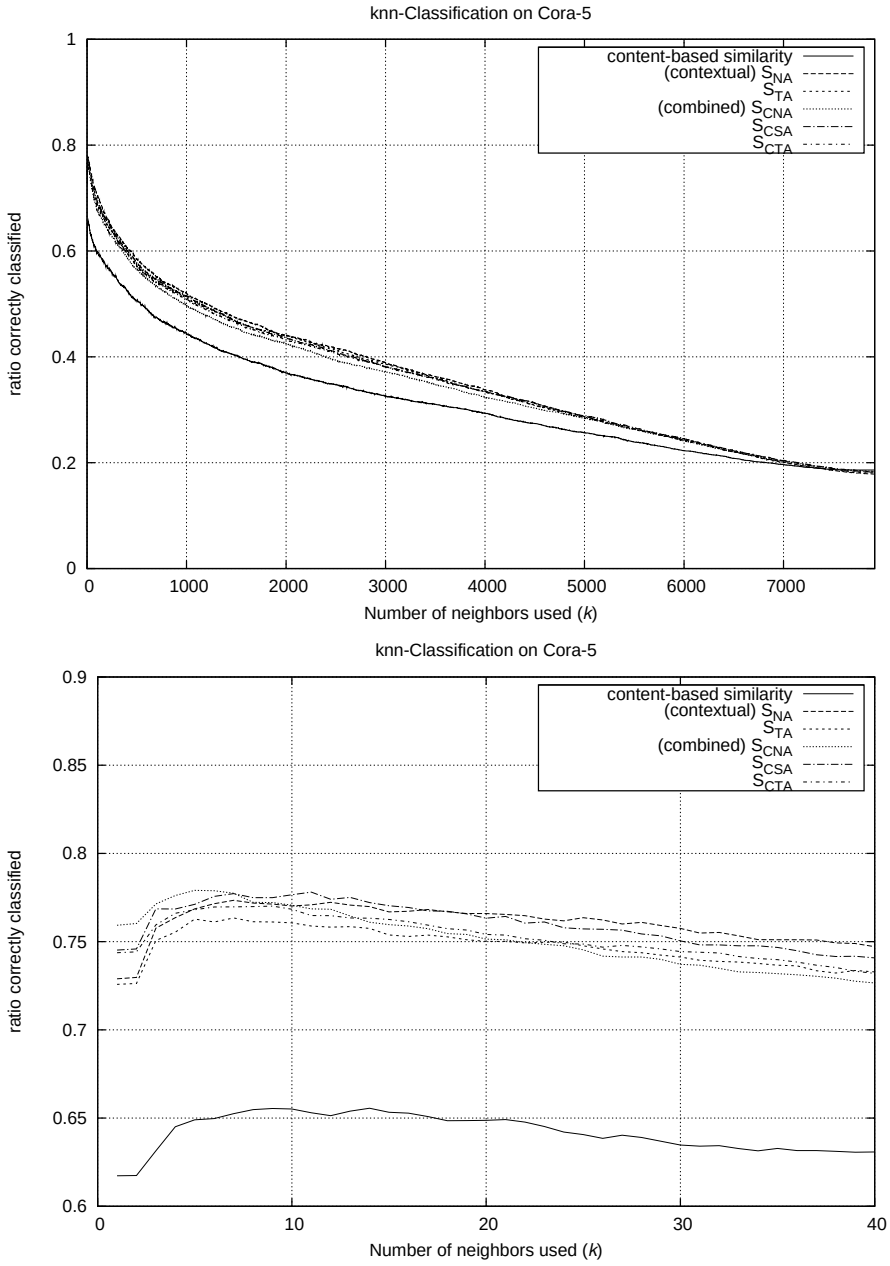


Figure 9.3: Ratio of correctly classified elements on CORA-5 with KNN-classification with k from 1 to 7916 (top) and zoomed in on k from 1 to 40 (bottom) for the five hybrid similarities compared to the original content-based similarity.

Agglomerative Hierarchical Clustering

Similarity	Cora-1	Cora-2	Cora-3	Cora-4	Cora-5
$\mathcal{S}_{content}$	0.58	0.44	0.42	0.44	0.38
$\mathcal{S}_{NA}$	0.63	0.58	0.54	0.50	0.45
$\mathcal{S}_{TA}$	0.63	0.58	0.54	0.50	0.46
$\mathcal{S}_{CNA}$	0.67	0.54	0.52	0.50	0.44
$\mathcal{S}_{CSA}$	0.64	0.58	0.54	0.49	0.46
$\mathcal{S}_{CTA}$	0.66	0.53	0.53	0.52	0.45

K-medoids

Similarity	Cora-1	Cora-2	Cora-3	Cora-4	Cora-5
$\mathcal{S}_{content}$	0.37	0.27	0.24	0.22	0.18
$\mathcal{S}_{NA}$	0.46	0.39	0.36	0.33	0.30
$\mathcal{S}_{TA}$	0.46	0.39	0.36	0.34	0.31
$\mathcal{S}_{CNA}$	0.46	0.37	0.34	0.32	0.28
$\mathcal{S}_{CSA}$	0.48	0.40	0.37	0.35	0.32
$\mathcal{S}_{CTA}$	0.48	0.40	0.38	0.35	0.22

knn-Classifiaction

Similarity	Cora-1	Cora-2	Cora-3	Cora-4	Cora-5
$\mathcal{S}_{content}$	0.83	0.75	0.68	0.69	0.66
$\mathcal{S}_{NA}$	0.93	0.88	0.82	0.82	0.77
$\mathcal{S}_{TA}$	0.92	0.87	0.82	0.80	0.76
$\mathcal{S}_{CNA}$	0.93	0.88	0.82	0.82	0.78
$\mathcal{S}_{CSA}$	0.93	0.88	0.82	0.81	0.78
$\mathcal{S}_{CTA}$	0.92	0.87	0.82	0.81	0.77

Table 9.2: Best results found for each clustering method, similarity, and subset. For agglomerative hierarchical clustering and k-medoids, this score is the best found F-score, and for knn-classification this score is the ratio of correctly predicted elements.

Part IV

Conclusions

Chapter 10

Conclusions

This chapter presents all the conclusions from this thesis.

10.1 Problem Setting

Data mining techniques are designed for a specific type of data. However nowadays, many datasets consist of a combination of different types of data. A data mining technique that is created for one such type can not use all the information in this dataset. This leaves the question of whether it is possible to create a technique that enables data mining techniques designed for one type of data, to also use the information from the other data type, albeit indirectly.

We proposed a method to create hybrid similarity measures. These are measures that alter a similarity measure based on the content of the elements, so that it also includes relational information from this dataset. Theoretical analyses suggest that these hybrid similarities use the homophily in the dataset to compensate for the content variability in it. The experiments described in Part II *Implementations* examine whether it is possible to implement the hybrid similarities succesfully. The experiments described in Part III *Analysis* examine what the reasons could be for the hybrid similarities to outperform the original, content-based similarity. In the following, we summarize the main conclusions from this research.

10.2 Conclusions from Implementations

The hybrid similarities easily can be implemented on data mining techniques that only use a similarity measure between elements (e.g. agglomerative hierarchical clustering, k-medoids, and KNN-classification). Here, the hybrid similarities indeed enhance the performance of the data mining technique when they are used on subsets of the well-known Cora dataset.

Implementing the hybrid similarities with k-means is not straightforward since a similarity between an element and a prototype for a cluster, which is outside the dataset, is not defined. However, it can be approximated by two newly proposed methods. These two methods boil down to using approximate similarity measures so that k-means becomes applicable. For these approximate versions of k-means, the following holds:

- Despite the fact that the hybrid similarities can only be applied to approximate versions of k-means, this setting still will lead to better results.
- Of these approximate versions of k-means, there is no significant difference in the quality of the found clusterings, but k-means-NAMA is much faster than k-means-NAM.
- The more sophisticated versions k-means-NAM(A) were worth developing, as they work better than the more straight-forward approach of applying k-medoids.

10.3 Conclusions from Analysis

The reason that the hybrid similarities improve the performance of these data mining techniques is indeed related to the fact that the homophily in the network compensates for the content varibility. The theoretical arguments for this, presented in Chapter 4, were confirmed experimentally in Chapter 8.

Experiments on a synthetic dataset show that the hybrid similarities consistently outperform the content-based similarity when the dataset has much homophily and much content variability.

Theoretical analyses show that there are five conceptual possibilities to combine the similarities from one element to the neighbors of the other in order to create a hybrid similarity. Experiments on subsets of Cora show for agglomerative hierarchical clustering, k-medoids and KNN-classification the following:

- The five hybrid similarities outperform the original, content-based similarity in every setting.
- There is no significant difference between the performance of any of the five hybrid similarities.

All in all, it can be concluded that, when there is sufficient homophily and content variability, replacing the content-based similarity with a hybrid similarity can lead to improved performance for data mining techniques that use a content-based similarity measure.

Bibliography

[1] R. Agrawal, T. Imielinski, and A.N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 1993.

[2] T.W. Anderson. *An Introduction to Multivariate Statistical Analysis*. John Wiley and Sons Ltd., 1958.

[3] S. Baraty, D.A. Simovici, and C. Zara. The impact of triangular inequality violations on medoid-based clustering. In *Foundations of Intelligent Systems*, volume 6804 of *Lecture Notes in Computer Science*, pages 280–289. Springer Berlin / Heidelberg, 2011.

[4] H. Blockeel and L. De Raedt. Lookahead and discretization in ILP. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 77–84, Berlin, 1997. Springer.

[5] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[6] R.E. Bohn and J.E. Short. How much information? 2009 report on American consumers. Technical report, Global Information Industry Center, University of California, San Diego, 2009.

[7] U. Bohnebeck, 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, Berlin, 1998. Springer.

[8] D. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.

[9] L. Breiman, J.H. Friedman, R. Olshen, and C.J. Stone. *Classification and Regression Trees*. Chapman and Hall, New York, 1984.

[10] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 255–264, Tucson, AZ, 1997.

[11] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[12] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[13] W.W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, 1995.

[14] T.F. Coleman and J. Moré. Estimation of sparse jacobian matrices and graph coloring problems. *SIAM Journal on Numerical Analysis*, 20(1):187–209, 1983.

[15] T.M. Connolly and C.E. Begg. *Database Systems: A Practical Approach to Design, Implementation and Management*. Addison-Wesley Publishing Company, 5th edition, 2009.

[16] D.J. Cook and L.B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[17] D.J. Cook and L.B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15(2):32–41, 2000.

[18] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *Knowledge Based Systems*, 8:373–389, 1995.

[19] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[20] S. Džeroski. Data mining in a nutshell. In *Relational Data Mining*, chapter 1. Springer-Verlag, 2001.

[21] C. Elkan. Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, pages 147–153, Washington DC, US, 2003.

[22] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, OR, 1996. AAAI Press.

[23] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34, Cambridge, MA, 1996. MIT Press.

[24] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[25] E. Fix and J.L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[26] K. Florek, J. Lukaszewicz, J. Perkal, and S. Zubrzycki. Sur la liaison et la division des points d'un ensemble fini. *Colloquium Mathematicae*, 2:282–285, 1951.

[27] T. Gärtner, P.A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop*, 2003.

[28] S. Geisser. *Predictive Inference*. Chapman and Hall, New York, 1993.

[29] M. Girvan and M. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[30] C. Goutte, L.K. Hansen, M.G. Liptrot, and E. Rostrup. Feature-space clustering for fMRI meta-analysis. *Human Brain Mapping*, 13(3):165–183, 2001.

[31] E.-H. Han, G. Karypis, and V. Kumar. Text categorization using weight adjusted k-nearest neighbor classification. In *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2001.

[32] J. Han, M. Kamber, and A. Tung. Spatial clustering methods in data mining: A survey. In *Geographic Data Mining and Knowledge Discovery*. Taylor & Francis, 2001.

[33] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Dallas, TX, 2000.

[34] D. Heckerman. Bayesian networks for data mining. *Data Mining and Knowledge Discovery*, 1:79–119, 1097.

[35] J. Heller. *Catch-22*. Vintage Books, 1994. First published in 1962 by Jonathan Cape Ltd.

[36] R.C. Holte. Very simple classification rules perform well on most commonly used data sets. *Machine Learning*, 11:63–91, 1993.

[37] M Honarkhah and J Caers. Stochastic simulation of patterns using distance-based pattern modeling. *Mathematical Geosciences*, 42:487–517, 2010.

[38] T. Horváth, T. Gärtner, and S. Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the 10th ACM SIGKDD Internatonal Conference on Knowledge Discovery and Data Mining*, pages 158–167, 2004.

[39] W.M. Huang and R.P. Lippman. Neural net and traditional classifiers. In *Neural Information Processing Systems*, pages 387–396, 1988.

[40] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference of Principles and Practice of Knowledge Discovery in Databases*, 2000.

[41] S.C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3), 1967.

[42] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[43] D.R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30, 1993.

[44] G.V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29:119–127, 1980.

[45] L. Kaufman and P. J. Rousseeuw. Clustering by means of medoids. In *Statistical Data Analysis Based on the L_1-Norm and Related Methods*, pages 405–416. Elsevier Science, 1987.

[46] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data, An Introduction to Cluster Analysis*. John Wiley and Sons Ltd., New York, 1990.

[47] M. Kirsten and S. Wrobel. Relational distance-based clustering. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 261–270, Berlin, 1998. Springer.

[48] 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, Berlin, 2000. Springer.

[49] I.R. Kondor and J.D. Lafferty. Diffusion kernels on graphs and other discrete structures. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 315–322. Morgan Kaufmann Publishers Inc., 2002.

[50] S. Kramer. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, Cambridge, MA, 1996. AAAI Press/MIT Press.

[51] S. Kramer. *Relational Learning vs. Propositionalization: Investigations in Inductive Logic Programming and Propositional Machine Learning*. PhD thesis, Vienna University of Technology, 1999.

[52] M. Kryszkiewicz and P. Lasek. TI-DBSCAN: Clustering with DBSCAN by means of the triangle inequality. In *Proceedings of the Seventh International Conference on Rough Sets and Current Trends in Computing*, pages 60–69, 2010.

[53] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM)*, pages 313–320, San Jose, CA, 2001.

[54] P. Langley, W. Iba, and K. Thompson. An analyses of Bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 223–228, San Jose, CA, 1992.

[55] A.N. Langville and C.D. Meyer. *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton University Press, 2006.

[56] B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 16–22, 1999.

[57] H. Lei, L.R. Tang, J.R. Iglesias, S. Mukherjee, and S. Mohanty. S-means: Similarity driven clustering and its application in gravitational-wave astronomy data mining. In *Proceedings of the International Workshop on Knowledge Discovery from Ubiquitous Data Streams (IWKDUDS 2007)*. Warsaw, Poland, 2007.

[58] D.D. Lewis. Naive (Bayes) at forty: The independance assumption in information retrieval. In *Proceedings of the Tenth European Conference on Machine Learning: ECML-98*, pages 4–15, 1998.

[59] Y. Lin, X. Shi, and Y. Wei. On computing pagerank via lumping the Google matrix. *Journal of Computational and Applied Mathematics*, 224:702–708, 2009.

[60] P. Lyman and H.R. Varian. How much information, 2003. Available at http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/.

[61] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[62] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press., 2008.

[63] K.V. Mardia, J.T. Kent, and J.M. Bibby. *Multivariate Analysis*. Academic Press, 1979.

[64] A. McCallum and K. Nigam. A comparison of invent models for naive Bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[65] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3:127–163, 2000.

[66] L.L. McQuitty. Elementary linkage analysis for isolating orthogonal and oblique types and typal relevancies. *Educational and Psychological Measurement*, 17:207–229, 1957.

[67] G.W. Milligan. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika*, 45:325–342, 1980.

[68] A.E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[69] J. Moody and C.J. Darken. Fast learning in network of locally tuned processing units. *Neural Computation*, 1:281–294, 1989.

[70] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[71] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352, San Mateo, CA, 1988. Morgan Kaufman.

[72] J. Neville, M. Adler, and D. Jensen. Clustering relational data using attribute and link information. In *Proceedings of the Text Mining and Link Analysis Workshop, Eighteenth International Joint Conference on Artificial Intelligence*, 2003.

[73] S. Nijssen, Y. Chi, R. Muntz, and J.N. Kok. Frequent tree mining — An overview. *Fundamenta Informaticae*, 66:161–198, 2005.

[74] J.S. Park, M.-S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 175–186, San Jose, CA, 1995.

[75] J.R. Quinlan. Discovering rules by induction from large collection of examples. In *Expert Systems in the Micro Electronic Age*. Edinburgh University Press, 1979.

[76] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[77] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufman Publishers, San Mateo, CA, 1993.

[78] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77 (2), pages 257–286, 1989.

[79] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1058–1063, San Mateo, CA, 1993. Morgan Kaufmann.

[80] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the Sixth International Workshop on Algorithmic Learning Theory*, pages 80–94, Berlin, 1995. Springer.

[81] C.R. Rao. *Advanced Statistical Data Analysis of Multivariate Observations*. John Wiley and Sons Ltd., 1952.

[82] S. Reeves and M. Clarke. *Logic for Computer Science*. Addison-Wesley, 2003.

[83] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall International, Inc., Englewood Cliffs, NJ, third edition, 2010.

[84] J. Shawe-Taylor and N. Cristianini. *Kernel MEthods for Pattern Analysis*. Cambridge University Press, New York, 2004.

[85] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[86] P.H.A. Sneath. The application of computers to taxonomy. *Journal of General Microbiology*, 17:201–226, 1957.

[87] P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy — The Principles and Practice of Numerical Classification*. W.H. Freeman, San Francisco, CA, 1973.

[88] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[89] T. Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons. *Biologiske Skrifter*, 5(4):1–34, 1948.

[90] A. Srinivasan. The Aleph manual. Technical report, Computing Laboratory, Oxford University, 2000. Available at `http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/`.

[91] B. Stein and O. Niggemann. On the nature of structure and its identification. In *25th Workshop on Graph Theory, Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[92] H. Steinhaus. Sur la division des corps matériels en parties. *Bull. Acad. Polon. Sci.*, 4(12):801–804, 1957.

[93] D. Sullivan. comScore: US has most searches; China slowest growth; Google tops worldwide in 2009. Technical report, Search Engine Land, 2009. Available at `http://searchengineland.com/...`
`...comscore-us-most-searches-china-slowest-34217`.

[94] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education Inc., Boston, MA, 2006.

[95] R.L. Thorndike. Who belong in the family? *Psychometrika*, 18(4), 1953.

[96] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[97] V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons Ltd., New York, 1998.

[98] J.H. Ward. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58:236–244, 1963.

[99] T. Witsenburg and H. Blockeel. A method to extend existing document clustering procedures in order to include relational information. In *Proceedings of the Sixth International Workshop on Mining and Learning with Graphs*, Helsinki, Finland, 2008.

[100] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[101] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, pages 721–724, Maebashi City, Japan, 2002.

[102] Y. Zhou, H. Cheng, and J.X. Yu. Graph clustering based on structural/attribute similarities. In *Proceedings of the VLDB Endowment*, pages 718–729, Lyon, France, 2009.

Nederlandse Samenvatting

In dit proefschrift staan twee Nederlandse samenvattingen. De eerste samenvatting is geschreven voor mensen zonder achtergrond in de informatica. Deze zal op een toegankelijke manier uitleggen wat er voor dit proefschrift onderzocht is en wat de resultaten zijn. Hierbij wordt vooral gefocust op de context van het onderzoek en minder op de technische details. De tweede samenvatting is geschreven voor mensen met een zekere achtergrondkennis binnen de informatica. Hier wordt de context als bekend beschouwd en ligt de nadruk op de gebruikte technieken.

Samenvatting voor Leken

Computers spelen een steeds grotere rol in ons dagelijks leven. Hun enorme opslagvermogen biedt veel mogelijkheden. Hoe kunnen we de opgeslagen data nog beter in ons voordeel gebruiken? Dit kan door naar interessante patronen in deze databestanden te zoeken. *Datamining* (het delven van data) is de tak binnen de informatica die zich hiermee bezig houdt.

Wat een interessant patroon is, hangt onder andere af van de gebruikersvraag. Verschillende vragen worden beantwoord door verschillende dataminingprogramma's. Neem een databestand met teksten van boeken. Een gebruiker wil de boeken die over hetzelfde onderwerp gaan bij elkaar verzamelen. Deze toepassing noemen we *clusteren*. Nog een voorbeeld: een databestand bevat eigenschappen van eiwitten waarbij voor een deel van deze eiwitten bekend is in hoeverre ze gebruikt kunnen worden voor de ontwikkeling van medicijnen. Onderzoekers naar geneesmiddelen willen de eigenschappen van eiwitten die geschikt zijn voor het maken van medicijnen vergelijken met de eigenschappen van eiwitten die niet geschikt zijn. Dit kan helpen bepalen of een onbekend eiwit een goede kans heeft aan de basis te staan voor de ontwikkeling van een medicijn. Dit staat bekend als *classificatie*. Weer een derde databestand houdt van klanten in een supermarkt bij wat ze per bezoek kopen. Voor de winkelier is het onder meer interessant om te weten welke artikelen vaak tegelijk gekocht worden om zijn winkel beter in te delen. Deze techniek staat bekend als *associatie analyse*.

Naast de gebruikersvraag, is het soort data van belang bij de ontwikkeling van een dataminingprogramma. Neem het eerder genoemde clusteren van boeken. Het is voor een computer niet makkelijk om te bepalen in hoeverre twee boeken over hetzelfde onderwerp gaan. Een mogelijke oplossing is om te kijken naar hoe vaak woorden in boeken voorkomen. Twee boeken die veel woorden gemeenschappelijk hebben, hebben een grotere kans over hetzelfde onderwerp te gaan dan twee boeken die relatief weinig woorden gemeenschappelijk hebben. In dit geval is het soort informatie *inhoudelijk* van aard; de boeken worden geclusterd op basis van hun inhoud.

Informatie kan echter ook *relationeel* van aard zijn. Een sociaal netwerk zoals Facebook kan je zien als een groot databestand. Hier is de belangrijkste informatie of er een relatie bestaat tussen twee mensen. Clusteren betekent hier dus het verdelen van gebruikers in groepen 'vrienden'. Het programma probeert clusters te maken waarbij er relatief veel verbindingen zijn tussen mensen in hetzelfde cluster, maar relatief weinig verbindingen van het ene cluster naar het andere. In dit geval is de informatie relationeel; de gebruikers worden geclusterd op basis van hun onderlinge relaties.

Een clusterprogramma dat werkt op basis van inhoudelijke informatie werkt heel anders dan een clusterprogramma dat werkt op basis van relationele informatie. In de loop der tijd zijn er voor beide soorten informatie veel verschillende clusterprogrammas ontwikkeld. Het kan echter zijn dat een databestand bestaat uit beide soorten informatie. Dit betekent dat een clusterprogramma dat werkt op basis van inhoudelijke informatie niets kan met de relationele informatie die ook beschikbaar is en vice versa.

Een voorbeeld ter verduidelijking: in een databestand met wetenschappelijke artikelen is per artikel een korte samenvatting van de inhoud bekend en ook is bekend welke artikelen door welk ander artikel geciteerd worden. De samenvattingen zijn de inhoudelijke informatie en de citaties zijn de relationele informatie. Stel dat er in de samenvatting van een bepaald artikel voornamelijk woorden staan die te maken hebben met het ene onderwerp, maar dit artikel citeert voornamelijk artikelen die gaan over het andere onderwerp. Bij welk onderwerp moet je dit artikel dan indelen? Een clusterprogramma dat slechts naar n soort informatie (inhoudelijk of relationeel) kijkt, zal makkelijk tot een oordeel komen, maar er bestaat dan een redelijke kans dat dit niet het juiste onderwerp is.

De methode die in dit proefschrift wordt beschreven, probeert beide soorten informatie te benutten om zo tot een beter resultaat te komen. Dit betekent dat in het ene geval de inhoudelijke informatie de doorslag krijgt en in het andere geval de relationele informatie, afhankelijk van welke het meest bepalend is. De methode is algemeen toepasbaar in dataminingprogrammas die ontworpen zijn voor inhoudelijke informatie. Dit betekent dat je de methode eigenlijk kan zien als een soort plugin die relationele informatie toevoegt in een bestaand dataminingprogramma dat zelf alleen kijkt naar de inhoud. De methode is

getest op diverse clusterprogrammas en op een classificatieprogramma. In alle gevallen kwam het tot betere resultaten dan zonder het gebruik van deze plugin. Er kan dus geconcludeerd worden dat het met deze methode mogelijk is om de resultaten van bestaande dataminingprogrammas te verbeteren.

Samenvatting voor Informatici

Het ontwerp van een dataminingalgoritme hangt van veel factoren af. Een belangrijke factor is het soort data dat er gebruikt wordt. Er zijn heel veel verschillende soorten en voor elk is er een grote verscheidenheid aan dataminingalgoritmes ontworpen. Wanneer een databestand echter meerdere soorten data bevat, zal de gebruiker moeten kiezen welke hij wel en vooral welke hij niet wil benutten. Een alternatief is het ontwikkelen van een nieuw dataminingalgoritme voor deze combinatie van datatypen. Het aantal combinaties van verschillende datatypen in een databestand is echter exponentieel in verhouding tot het aantal soorten data en het is derhalve ondoenlijk om voor elke combinatie aparte dataminingalgoritmes te ontwikkelen.

In dit proefschrift onderscheiden wij twee belangrijke categorien informatie: inhoudelijke en relationele. Inhoudelijke informatie zegt iets over hoe een element eruit ziet en relationele informatie zegt iets over hoe een element zich verhoudt tot een ander element. In een geannoteerde graaf kan je de annotatie van de knoop beschouwen als de inhoudelijke informatie en de takken tussen de knopen als de relationele informatie. De methode die wij in dit proefschrift presenteren, voegt indirect relationele informatie toe aan een algoritme dat is ontworpen voor inhoudelijke informatie. Zo hoeft er dus niet voor elke combinatie van een inhoudelijk en een relationeel datatype in hetzelfde databestand een nieuw dataminingalgoritme ontworpen te worden.

De methode die in dit proefschrift gepresenteerd wordt, is in principe toepasbaar op elk dataminingalgoritme dat gebruik maakt van een afstands- of gelijkaardigheidsmaat op basis van inhoudelijke informatie. Onze methode kan dan gezien worden als een plugin die deze maat vervangt voor een die ook de relationele informatie in ogenschouw neemt. Dit wordt gedaan door voor de afstand of gelijkaardigheid tussen twee knopen ook de annotaties van de buren van deze knopen te gebruiken. Zo wordt de relationele informatie dus indirect gebruikt om een nieuwe maat te genereren.

Een goede vergelijkingsmaat is belangrijk voor een goed resultaat. In de praktijk is het echter heel moeilijk om een goede maat te ontwerpen. Het kan dus zeer goed voorkomen dat zon maat in sommige gevallen een verkeerde uitkomst genereert wat er vervolgens voor kan zorgen dat een element in het verkeerde cluster wordt ingedeeld. Onze methode zorgt er voor dat een dataminingalgoritme robuuster beter bestand is tegen dit soort fouten.

Als nu de inhoud van een knoop niet zo erg lijkt op de inhoud van knopen uit zijn eigen categorie, is het voor een vergelijkingsmaat moeilijk om de juiste

waarde voor die knoop in verhouding tot andere knopen te bepalen. Hierdoor wordt deze knoop waarschijnlijk verkeerd ingedeeld. In veel databestanden met relationele informatie zijn het vooral knopen die in dezelfde categorie horen die met elkaar verbonden zijn. Dit betekent dat over het algemeen de buren van een knoop voor het grootste gedeelte uit dezelfde categorie komen. De kans dat alle inhouden van de buren sterk afwijken van de gemiddelde inhoud van knopen uit die categorie is zeer klein. In dit geval geven de buren van deze knoop een betere representatie van de categorie van deze knoop en dus zorgt het feit dat onze methode gebruik maakt van deze buren ervoor dat de resultaten beter zullen zijn.

Onze methode is getest voor diverse clusteringalgoritmes en voor een classificatiealgoritme. In alle gevallen waren de resultaten beter dan het originele algoritme. Hieruit kan geconcludeerd worden dat deze methode in staat is om bestaande dataminingalgoritmes te verbeteren.

Het proefschrift bestaat uit vier delen. Het eerste deel behandelt de theoretische basis. In hoofdstuk 2 wordt de stand van zaken in data mining, voor zover relevant voor dit proefschrift, besproken. Hoofdstuk 3 beschrijft uitvoerig hoe de methode werkt. Ten slotte geeft hoofdstuk 4 een intutieve motivatie waarom onze methode voor betere resultaten kan zorgen.

Het tweede deel beschrijft de experimenten die gedaan zijn om de praktische toepasbaarheid aan te tonen. In hoofdstuk 5 wordt de methode toegepast op agglomerative hierarchical clustering. Hoofdstuk 6 wordt dit vervolgens gedaan voor k-means en k-medoids. Dit hoofdstuk laat tevens zien waarom deze methode niet direct kan worden toegepast op k-means en wat er kan worden gedaan om onze methode hier toch te gebruiken. In hoofdstuk 7 wordt deze methode toegepast op KNN-classification.

Het derde deel analyseert de werking van onze methode. In hoofdstuk 8 worden de effecten van ruis op de inhoud en ruis op de relaties met elkaar vergeleken. Hier wordt aangetoond dat de *homophily* in een geannoteerde graaf kan compenseren voor de *content variability*. Hoofdstuk 9 bespreekt welke varianten op onze methode mogelijk zijn en hoe deze ten opzichte van elkaar presteren.

Het vierde deel presenteert in hoofdstuk 10 tot slot de conclusies die uit alle onderzoeken kunnen worden getrokken.

Curriculum Vitae

Tijn Witsenburg is geboren in Leiden op 13 augustus 1975. Van 1987 tot 1994 doorliep hij achtereenvolgens het HAVO en VWO op het Fioretti College te Lisse. In 1994 begon hij aan de studie Natuur- en Sterrenkunde aan de Universiteit van Amsterdam, maar in 1995 stapte hij over naar de Universiteit Leiden, alwaar hij de dubbele propedeuse Wiskunde/Informatica ging doen. Van 2000 tot 2001 zat hij in de EL CID, de commissie die voor de Universiteit de introductieweek voor eerstejaars studenten organiseert. In 2004 studeerde hij in het kader van het Erasmus-project zeven maanden aan de Università Ca Foscari Venezia in Italië, waarna hij in 2006 aan de Universiteit Leiden afstudeerde in de Informatica. Vanaf 2007 was hij verbonden aan het Leiden Institute of Advanced Computer Science als promovendus, waar hij zijn promotieonderzoek uitvoerde binnen het NWO-project “Annotated Graph Mining”. Eveneens assisteerde hij diverse docenten bij hun onderwijsactiviteiten.

KLEURPLAAT