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

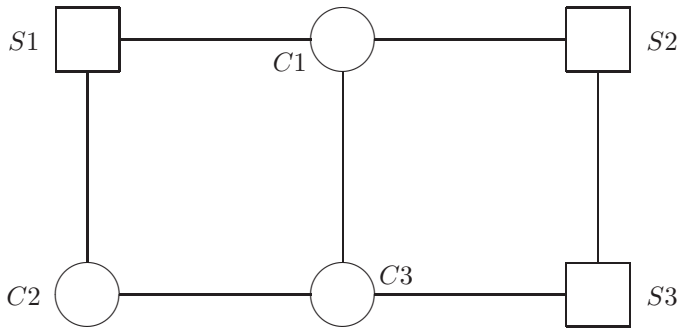

Figure 2.1: Example of a graph with two different kinds of vertices.

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.

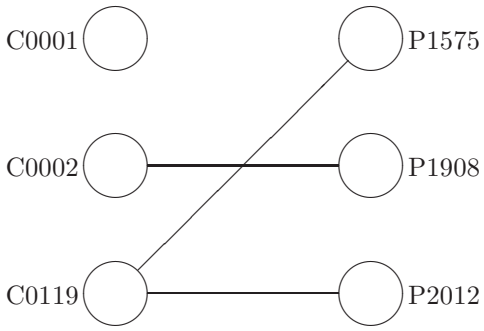

Figure 2.2: The structure of the relational database described in Tables 2.1 through 2.3 drawn as a graph.

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,

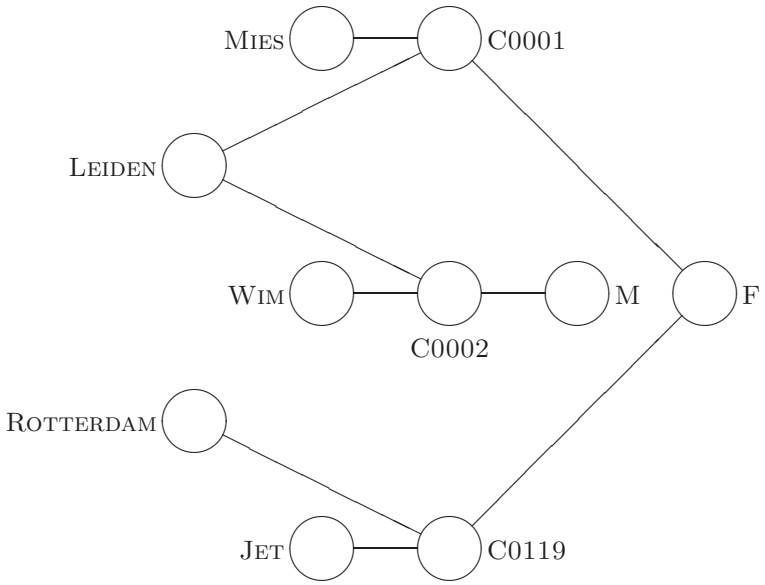

Figure 2.3: The relational database described in Table 2.1 drawn as a graph where each attribute value is a vertex.

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.