UNIVERSITY OF TWENTE.

Outcome and variable prediction for discrete processes

A framework for finding answers to business questions using (process) data

Master Thesis of Sjoerd van der Spoel
Business Information Technology,
Enschede, January 2012

Supervisors University of Twente
Maurice van Keulen
Chintan Amrit

Supervisors Topicus FinCare
Jasper Laagland
Johan te Winkel

Summary

The research described in this paper is aimed at solving planning problems associated with a new hospital declaration methodology called DOT. With this methodology, that will become mandatory starting January 1st 2012, hospitals will no longer be able to tell in advance how much they will receive for the care they provide. A related problem is that hospitals do not know when delivered care becomes declarable. Topicus Fincare wants to find a solution to both these problems.

These problems, and more generally the problem of answering business questions that involve predicting process outcomes and variables is what this research aims to solve. The approach chosen is to model the business process as a graph, and to predict the path through that graph, as well as to use the path to predict the variables of interest. For the hospital, the nodes in the graph represent care activities, and the variables to predict are the care product –that determines the value of the provided care – and the duration of care.

A literature study has found data mining and shortest path algorithms in combination with a naive graph elicitation technique to be the best way of accomplishing these two goals. Specifically, Random Forests was found to be the most accurate technique for predicting path-variable relations and for predicting the final step of a process. The Floyd-Warshall shortest path algorithm was found to be the best technique for predicting the path between two nodes in the process graph.

To test this findings, a number of experiments was performed for the hospital case. These experiments show that Random Forests and the Floyd-Warshall algorithm are indeed the most accurate techniques in the test. Using Random Forests, the care product for a set of performed activities can be predicted with on average 50% accuracy, lows of 30% and highs of 70%. Using Floyd-Warshall, the consequent set of steps can be predicted with 45% accuracy on average, with lows of 25% and highs of 100%.

From the experiment with the hospital data, a set of processing steps for producing an answer to a business question was produced. The steps are transforming the business question, analyzing and transforming data, and then depending on the business question classifier training and variable prediction or process elicitation and path prediction. The final step is to analyze the result, to see if it has adequately answered the question. That these processing steps do actually work was validated using a dataset from Topicus' bug tracking software. In conclusion, the approach presented predicts the total cash flow to be expected from the provided care with average error between six and 17 percent. The time the provided care becomes declarable cannot be accurately predicted.

Preface

The thesis you have in front of you now is the result of the work I have performed at Topicus Fincare between the end of July and the end of November 2011. It has been hard work at some times, but mostly I have really enjoyed writing this thesis and performing the research underlying it. Immediately when I started, I felt like the assignment I was provided with at Topicus was interesting and that my research could be useful, both for practical as for scientific purposes. This usefulness was of course for me to prove through research.

Over the course of doing the research I have run into some obstacles, mostly when it came to tools that would support the requirements for them. Even the best tool I could find –the R language– would often not cooperate. Still, this did mean that not only did I learn about the topic of this research, but that I got to learn another programming language, that has plenty of good uses. Besides data mining and R this research has also made me well acquainted with LaTeX for typesetting this report and Ti*k*Z for creating the images.

I would like to thank Topicus Fincare for giving me the opportunity to perform this research, and for giving me the resources needed to perform the research. Also, my thanks go out to my supervisors both at Topicus and at the University of Twente for giving me feedback when I asked for it and generally for helping me along the way. I hope you enjoy reading this thesis as much as I have enjoyed writing it.

Contents

Chapter 1

Introduction

Every company faces uncertainty as to what the result of a business process will be and when that result will be achieved. This is a fact of life when doing business. Not knowing what cash flows will start at what time makes it difficult to make plans, such as decisions on when to invest.

For hospitals, not knowing how much outstanding income they have is even more of a problem, as insurance companies demand hospitals to give these figures twice a year.

Starting January 1st, 2012, Dutch hospitals will use a new system for claiming the costs they make for treating patients. Hospitals claim these costs at patient's insurance companies, who then bill their customers. Topicus, a Dutch software company, provides solutions for the health care industry. Currently, Topicus is developing a solution to support the administrative processes (registration, claiming) that underly this new system. To explain how this thesis can contribute to Topicus' administrative solution, this section describes the current and new hospital claiming and registration system, respectively called DBC and DOT.

1.1 Hospital claiming methodology

1.1.1 Current system

The registration and claiming methodology currently in place (mandatory for all care providers) is called Diagnosis-Treatment Combination (in Dutch: Diagnose Behandelcombinatie, DBC). The DBC system revolves around a diagnosis, to which several treatments are prescribed. The specific treatments (in DBC terminology: activities) that are performed to treat a patient are registered as part of the DBC.

A downside of the DBC system is that it uses averages for the amount and type of treatments associated with a diagnosis. If a patient requires less treatments than are registered to the DBC, the hospital can still declare the price of the opened DBC. The other way around is also possible, where a patient requires more treatments than are registered to the set diagnosis. This is dealt with by opening a new DBC, starting when the original DBC was closed. Another problem is that the DBC is not specific in the type of treatment, where

there are sometimes more and less expensive options to treat the same diagnosis.

Besides these more finance or claim registration related problems, the two largest problems of the DBC system (and main reasons to redesign the system as mentioned further on in this section) are:

- There are 300.000 DBCs in the current system. Consequently, negotiating with insurers over the price of these DBCs is time consuming. Also, medical specialists have difficulty in recognizing the actual medical condition which is covered by a DBC.
- The DBC methodology does not take days a patient spends in the hospital into account.

The consequence of the problems above is that the actual costs of treating a patient are often not equal to what the hospital charges the insurance company for treating that patient's diagnosis.

1.1.2 DOT system

The new registration and claiming system addresses the problem (actual costs are unequal to what is charged) stated above in an effort to increase the transparency of cost calculation for provided care. The system is referred to as DOT, which stands for "DBCs towards transparency".

The DOT system takes a different paradigm to DBCs: the central principle of DOT is to decide on care product based on the care provided, whereas in the DBC system the care product is based solely on the diagnosis.

A care product has (in both DBC and DOT) an associated price, which is claimed at a patient's insurance company. To derive which care product a patient has received, the DOT system uses a system called the *grouper*. This grouper consists of rules that specify how care products are derived from performed activities. DOTs are processed by the grouper after they have been closed. When a DOT is closed depends on the amount of time that has passed since the last activity. If more than the specified number of days has passed, the DOT is marked closed. Different types of activities have different durations after which the DOT is to be marked as closed.

The DOT system should lead to a better matching between actual provided care and associated care product. In turn, this leads to what the insurance companies (and therefore patients) pay matching the care received by the patient. Two main problems arise with the DOT system, of which the second one is also part of the DBC system:

- The hospital does not know how much they will receive for the care they have provided, as they don't know what care product will be associated with the open DOTs.
- The hospital does not know when a DOT is likely to be closed, because a DOT closes only some time after the final treatment. If the patient turns out to require another treatment before the closing date of the DOT (based on the previous treatment), the closing date moves further backward. Because the hospital does not know when a DOT closes, they also don't know when they can declare the associated care product.

Chapter 2

Goals

2.1 Business goals

Based on the previous two sections on the DBC and DOT system, we can describe what goals Topicus has for this project. As mentioned at the start of the background section Topicus is developing a software solution that supports the DOT system. This solution will be used to register the diagnosis, the performed activities and will support the process of claiming closed sub paths. Part of this product is a financial reporting solution, for which it is necessary to be able to estimate the expected size and expected time of cash flows. More specifically, Topicus FinCare wants an answer to the question *"When can the hospital expect cash flows for its treatment processes and how large will those cash flows be?"*.

Topicus' goal for this project is therefore to design, implement and test a system that answers this question.

2.2 Research goals

The problem presented by Topicus is that hospitals don't know the value for a process variable (the "work in progress" or expected cashflow is unknown). A different formulation of the business problem is that the relation between the process and the variable "duration" or "value" is not known. So generally, it is a problem of not knowing what steps make up the process and how some process variable can be predicted from that process. This problem class is what the research goal aims to solve. Solving the problem class will mean that the problem instance (the hospital case) is solved as well.

The research goal is therefore to find a general way to answer a business question that is related to a (business) process. Specifically, that process is either unknown, or is (implicitly) different from its design, and the business question is about wanting to know how the process affects a variable, or how a process variable affects another process variable. The business question is answered using information from the past, so the past is used to predict the future.

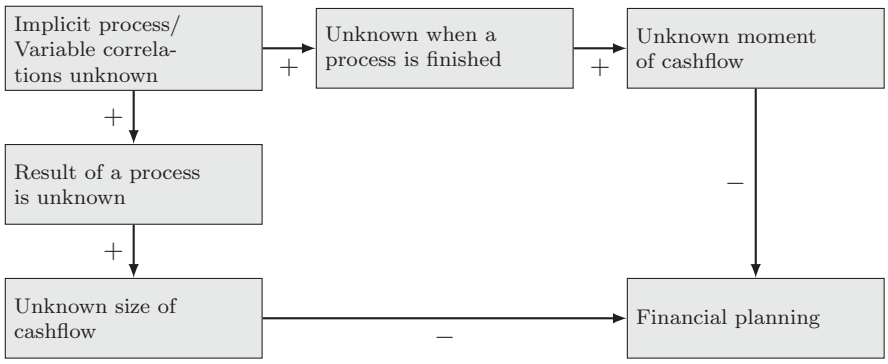


Figure 2.1: Causality graph for the problem statement

This goal separates into three parts:

- *finding the process where it is implicit.* This means that we want some way to discover the process as it actually happens/as it can be observed. This also applies to situations where there is an existing process design or set of process rules, because the designed process might well differ from the observable *actual* process.
- *finding relations between process and variables.* To answer the business question, such as in the hospital case, some rules have to be induced on the data set, that describe the relationship between the variables of interest (how one affects the other). From now on, this will also be referred to as *correlation* (some dependency or influence of one thing on another) *induction* (retrieve from data).
- *finding a general way.* Having the above two ingredients, a complete description will be made of a data-centered approach for answering a business question (finding relations) related to a(n) (implicit) business process.

To clarify the relation between the case goal and the research goal – or problem and problem class – figure 2.2 shows how the concepts from the problem (hospital case) map to those of the problem class. The upper half shows an example care (sub) path and how its elements translate to the general class. The bottom half op the figure shows an ontology of the hospital case concepts and of the problem class concepts and how they map to eachother. Chapter 3 (Concepts) gives a more detailed formal description of these concepts.

2.3 Problem statement

This section describes the gap between current situation and the goal situation. A goal is a desired situation, the gap between current and desired situation is the definition of the problem. The current situation has been described in section 1.1. The gap that exists between current and goal situation can be further refined through a causality analysis, that shows what causes the gap. This is a financial problem, more specifically one of financial planning (knowing what to expect).

The causal graph in Figure 2.1 shows the interrelation of the underlying causes for the financial planning problem. A + sign in the graph means that the node at the start of the edge influences positively (increases) what is mentioned in the node at the end of the edge. The minus sign has the opposite meaning: one node has a negative effect on the other.

"Unknown when a process is finished" means there is no way to be sure how many steps there will be until the end of a process and how long those steps will be. This only applies to those processes that are expected to end at some time in the future.

"The result of a process is unknown" refers to the situation where processes are classified once they are finished. The process result could for instance be classified as "Large sale", "Big project" and of course, in the DOT case, as a care product. Since we can't be sure *what* the outcome of the process is, we can't be sure what the corresponding cash flow (if any) will be either. Since we don't know *when* the outcome of the process is achieved, we don't know when that cash flow will start. If we don't know what to expect, we cannot perform meaningful financial planning.

If we know the process and know which step we're in now, we know which steps we will likely take. Then we know which cash flows are likely to result. If we know the relation between the process steps and duration, we know how to estimate the process duration.

2.4 Summary

This section has described Topicus' business goal for this project, which is to solve the hospital's problem of not knowing when to expect what cash flows. Solving the problem class this belongs to is the associated research goal: answering a business question related to an implicitly defined process.

The current situation is that a hospital cannot predict the care products that belong to unfinished sub paths. Hospitals can find the care product of a *finished* sub path through the grouper. The gap with the desired situation is bridged by a tool or model that can explain or predict what the subsequent activities are. Furthermore, the current situation is that a hospital cannot predict the duration of a sub path. The hospital does know how long an activity (averagely) takes. Therefore the same tool or model might be used to predict the duration from its prediction of subsequent activities, bridging this gap.

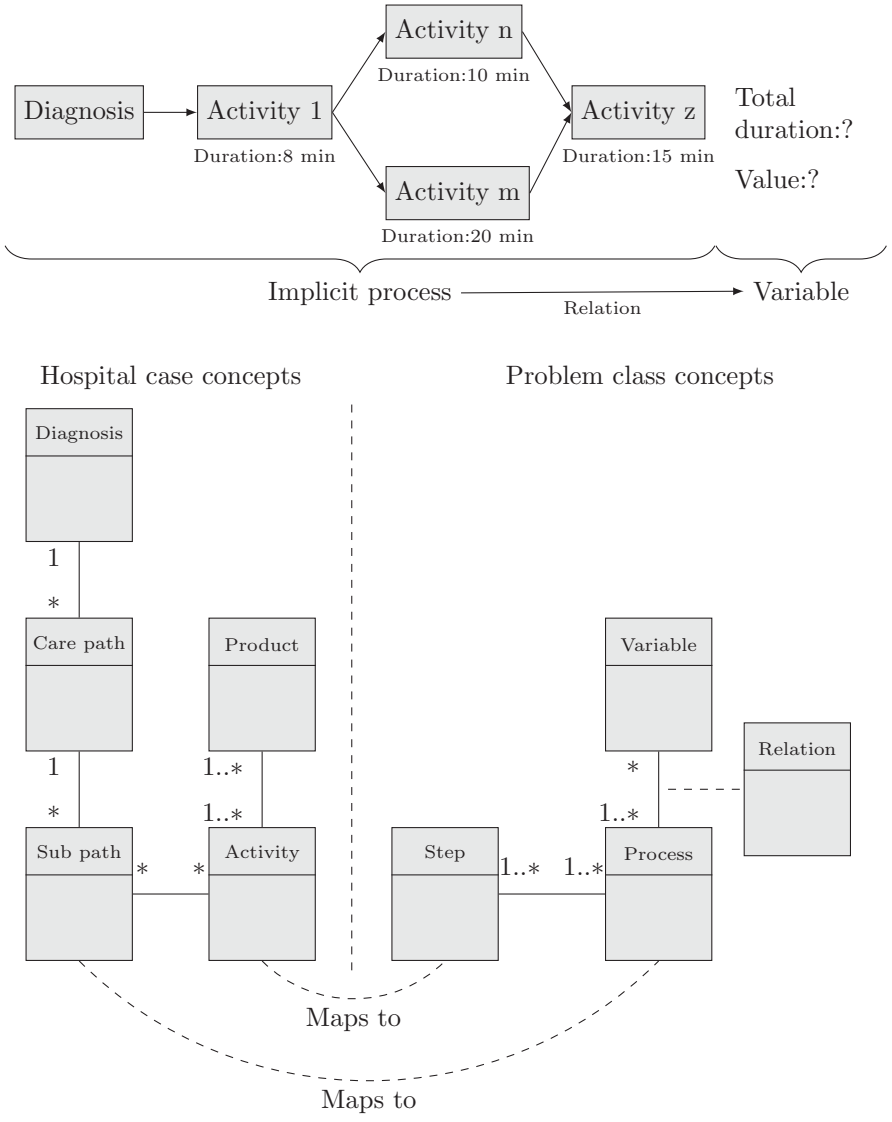


Figure 2.2: Mapping the case problem to the problem class

Chapter 3

Concepts

The contribution of this chapter is threefold. First, we want to make clear what the problem is that the research addresses by operationalizing it, since the more clear it is what the problem means, the more clear it is when we have solved the problem. Operationalizing the problem means making it and its aspect measurable. Second, we want to make clear what the operational goal is, so it will be more clear when we have achieved that goal. Also, an operational goal makes it possible to test if we have achieved that goal. Finally, this chapter will make clear what we mean by some of the other concepts brought up in the research. The previous chapter has brought up a number of concepts, both relating to the hospital case and the class of problems that case belongs to. This chapter gives operational definitions and descriptions of those concepts that are recurring elements in this thesis.

3.1 Problem operationalization

The main concept discussed in the introduction of this thesis is the hospital problem: they don't know what cashflow to expect at what time. We have already identified (through figure 2.1) that this is caused by

1. a lack of understanding of the process
2. a lack of understanding of how the process steps taken affect process variables

To operationalize these problems – that are about processes – we need a formalization of the process. A process consists of steps and rules for moving from one step to another step (the rules dictate what the possible next steps are after completing one step). These steps and rules can be expressed as a graph, where the steps are nodes and the rules are edges in the graph. This notion is shared by a common process description language: Petri nets [16].

We therefore define a process as a directed graph with a set of nodes N and set of edges E, one starting node n_{start}, at least one end node n_{end} such that $\{e(n_{end}, n)|n \in N - n_{end}\} = \varnothing$ (it has no outgoing edges) and $\{e(n, n_{start})|n \in N - n_{start}\} = \varnothing$ (it has no incoming edges), where $e(n, n')$ denotes an edge from n to n'. This definition also applies to the hospital's sub path concept.

The lack of understanding means that neither does the hospital know what this graph is, nor does it know what path will be taken through that graph. A path is defined as a set of edges:

$$\{E_{path} = \{e_1, \ldots, e_n\} \subseteq E \mid e_1.start = n_{start} \ \wedge e_n.end = n_{end} \ \wedge \\ \forall_{1<i<n} \ e_i.start = e_{i-1}.end \ \wedge e_i.end = e_{i+1}.start\}$$

Note that our definition of a path is identical to that of an *open walk* in graph theory: its start and end node are different and it may contain cycles. For our purposes, we narrow the definition to only those open walks that start at the start node and end at the end node. For the hospital case, the set of activities (a path through the process graph) is used to determine the care product that was performed. This is done by feeding the path and the diagnosis into the grouper: a binary directed graph that is traversed, where each choice in the tree is made based on the set N_{path} containing a certain activity. The leaves of the tree are the care products.

The hospital wants to know the care product, before the path is finished. This problem is caused by the hospital not knowing the process graph and not knowing the path to the end node, given a path from the start node. For example, the activity being undertaken at moment x is D in figure 3.1. Suppose we have had a path $A - B - D$ to this node, we want to know which edges will be traversed to get to H. Every subsequent path ($D - H$, $D - E - G - H$ or $D - E - F - G - H$) will possibly lead to a different outcome (care product) from the grouper. Given the formalization of the problem above, the hospital's business questions are formalized to: what will be the path through the graph? What is the duration of that path?

3.1.1 Definition of the problem class

The hospital has a *business question* that is related to their *business process*. They do not know *explicitly* what this process is. If there is a process design, they do not know if the *actual process* conforms to that design. The business question is that they do not understand the mechanics of the relation of this process and two variables: value of the care product and duration of the sub path.

In abstract, the problem is: a business does not know *explicitly* what its *actual* business process is. This means they do not know which process steps will follow after some set of initial steps and/or they do not know how this influences process variables – like duration. The process is not continuous, like a production line, but has an input and output – it is *discrete*. It is about delivering a *service*, not a product. This is the problem class: *business questions from service oriented organizations that are related to their discrete and implicit business process.*

3.2 Goal operationalization

The previous sections have described the problem in an operational sense. This section takes this definition of the problem and uses it to create an operational definition of the goal. The goal of the research is to mitigate or resolve the

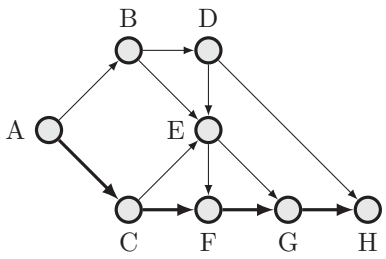


Figure 3.1: A process graph

problem. Since not knowing the process graph and expected future path through that graph is the problem, the goal is to find a way to establish the process graph and to predict the path. There are two things that need to be achieved for the hospital case: a way to find the process graph and a way to find what the most likely path is through that graph. An example process graph is shown in figure 3.1, where the thick edges represent the most likely path. These two techniques will compose the framework, together with a way of finding path/process related relations. We will refer to both finding relations that aren't directly process associated and to finding the most likely path through the process graph as correlation induction. Finding the process graph is what we refer to as process elicitation.

The accuracy with which we can establish the value for these variables therefore depends on the process graph and the technique for finding a path through that graph. This means it is important to specify how accurate the process graph and pathfinding technique should be. The needed accuracy will differ for every case, but the framework should at the very least give an accuracy estimate – such as a confidence interval.

3.2.1 Definition of research concepts

Below, we will give a more formal definition for the elements mentioned above: correlation, correlation induction, process elicitation, accuracy, processing time and business question. This formalization is used to make these concepts measurable, it is therefore an operationalization.

- *Business question* A business question is a knowledge problem related to some (sets of) data variables. Examples include a business wanting to know what the effect will be on variable B if they change variable A, or a business wanting to predict their revenue for the next month, based on previous monthly figures. These questions are about predicting a future value for a variable A or predicting variable change. Formally, this means predicting A_{t+1} from A_t or predicting ΔB from ΔA. To produce these results, the relation $A_t \xrightarrow{?} A_{t+1}$ and the relation $A \xrightarrow{?} B$ must be explored.

  The nature of the variables and the expected nature of the relation help decide which approach should be chosen to find an answer. It might well be the case that both correlation induction and process elicitation is needed to understand the relation.

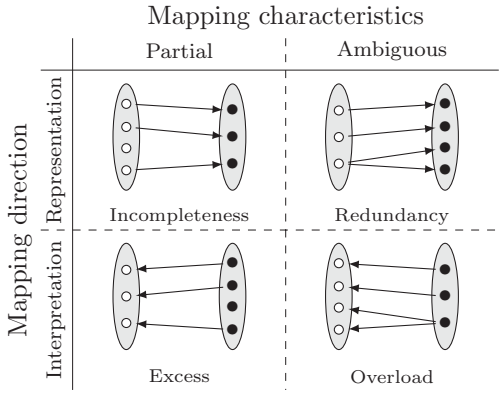


Figure 3.2: Mapping ontological to grammatical constructs [21]

- *Correlation* Two variables or datasets A and A' are said to have a correlation if the value/composition of A is dependent on the value/composition of A' –or vice versa. Dependency means that there is a statistical relationship between A and A'. There is a dependency if specific pairs of values/ compositions of A and A' are found more than some threshold amount. An example threshold is statistical significance.
- *Process elicitation* refers to a procedure for finding a process graph. A process elicitation procedure takes a set of activities $A = \{A_1, \ldots, A_n\}$ and constructs a graph with set of nodes $N = A$ and set of edges $E \subseteq N \times N$.
- *Completeness* is

  a. For process elicitation: The extent to which a process model reflects the observable–actual– process. The comparison of model and actual is part of the field of ontological comparison, where it is called the fit of ontology (model) and grammar (actual situation). Green and Rosemann [21] describe four measures for two metrics: ontological clarity and ontological completeness. These measures are shown in figure 3.2.

  A white circle represents an element of the model, a black circle represents an element of "the real world". A perfect model will have none of the characteristics in the figure, meaning that every node in the process graph represents exactly one node in the actual process, every node in the actual process is represented by exactly one node in the graph, and similarly for edges.

  The model is ontologically complete and clear iff $\forall n.\{n \in N_{actual} \leftrightarrow n \in N_{model}\} \wedge \forall e.\{e \in E_{actual} \leftrightarrow e \in E_{model}\}$. This leads to the two completeness measures c_{nodes} and c_{edges}:

$$c_{nodes} = \frac{|\{N_{model} \cap N_{actual}\}|}{|N_{actual}|}$$

$$c_{edges} = \frac{|\{E_{model} \cap E_{actual}\}|}{|E_{actual}|}$$

The compound completeness measure is given by:

$$completeness_{model} = \min(c_{nodes}, c_{edges})$$

b. For correlation induction: The extent to which the relations found reflect the correlations that could be found. Completeness is given for a set of found relations R and a dataset D by:

$$\frac{|\{(r_1, \ldots, r_n) \in R \mid dependency(r_1, \ldots, r_n) > threshold\}|}{|\{(d_1, \ldots, d_n) \in D \times D \times \cdots \times D \mid dependency(d_1, \ldots, d_n) > threshold\}|}$$

The dependency function will be different for the type of relations that are induced: if they are numeric ($A = 0.3 \rightarrow B = 0.5$) the function will be statistical (such as regression analysis), if the relations are based on sets ($A \subseteq X \rightarrow B \subseteq X$), the dependency function will use a different measure. Completeness is maximized if all dependencies above a threshold present in the dataset are present in the set of found relations.

- *Accuracy* is

a. For process elicitation: The extent to which the elements of the process model are found in the actual model. Given a set of nodes in the model N_{model}, the actual set of nodes N_{actual}, a set of edges in the model $E_{model} \subseteq N_{model} \times N_{model}$ and an actual set of edges $E_{actual} \subseteq N_{actual} \times N_{actual}$, we can give two measures of accuracy, one for edges, one for nodes:

$$accuracy_{nodes} = \frac{|\{n \in N_{model} \mid n \in N_{actual}\}|}{|N_{model}|}$$

$$accuracy_{edges} = \frac{|\{e \in E_{model} \mid e \in E_{actual}\}|}{|E_{model}|}$$

The compound accuracy of these two measures is given by:

$$accuracy_{model} = \min(accuracy_{nodes}, accuracy_{edges})$$

b. For correlation induction: The extent to which the relations found are actually relations in the dataset: Given a dataset D and a produced set of correlations $R \subseteq (D \times D \times \cdots \times D)$,

$$accuracy = \frac{|\{(r_1, \ldots, r_n) \in R | dependency(r_1, \ldots, r_n) > threshold\}|}{|R|}$$

Just as mentioned before, the type of correlation dictates the dependency measure. Accuracy is maximized if all correlations found have a dependency of at least the threshold value.

3.2.2 Example of the performance metrics

The meaning of the completeness and accuracy measures for process graphs are shown in figure 3.3. It shows a one hundred percent accurate (and therefore complete) sets of nodes and edges, as well as one incomplete and inaccurate graph. The accuracy for that graph's edges is $\frac{2}{9}$, for its nodes it is $\frac{5}{6}$. This means the compound accuracy is $\frac{2}{9}$. The completeness for nodes is $\frac{5}{7}$, for edges it's $\frac{2}{9}$. Therefore, compound completeness is $\frac{2}{9}$ as well.

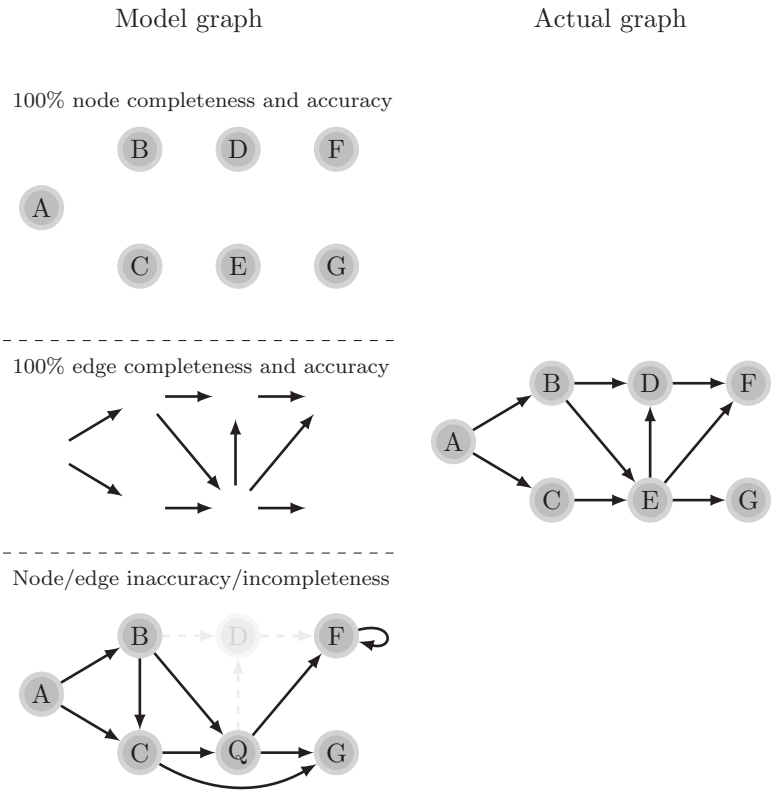


Figure 3.3: Overview of formal concepts

Chapter 4

Research setup

In chapter 2, we found that the hospital problem is member of the "implicit process" class of problems. More precisely, the hospital problem can be generalized as "the process is defined implicitly –or not at all– and its effect on process variables is unknown". This problem is solved if we find a tool that can predict process variables and/or a methodology to make the actual process explicit. This is because –as mentioned in chapter 3– knowing the process graph, knowing the path we will likely take through that graph and knowing the effect of every activity (node) on a variable means knowing the compound effect of the activities in the path on the variable.

The question is: how do you find the process graph and how do you find the most likely path through that graph? The constraint here is that the solution must be applicable to all instances in the problem class. The problem class is described in detail in section 3.1.1.

4.1 Research question

The solution concept for the problem class (research goal) is illustrated in figure 4.1. It shows a business question and a situation conforming to the constraints (instance of the problem class) denoted by elements in the brackets. Within the "cloud", some processing is done, resulting in something that answers the business question, be it a description of the correlation of variables, a tool for predicting future process steps or just the business process made explicit (or all of these). What this research aims to contribute is the processing steps that happen within the cloud. Specifically, we will research the following set of questions:

I. What methods exist for finding a process graph and which is the most accurate?

II. What methods exist for finding the most likely subsequent path through a process graph, given a set of previously traversed nodes, and which of those is the most accurate?

III. What methods exist for finding path-variable correlations and which is the most accurate?

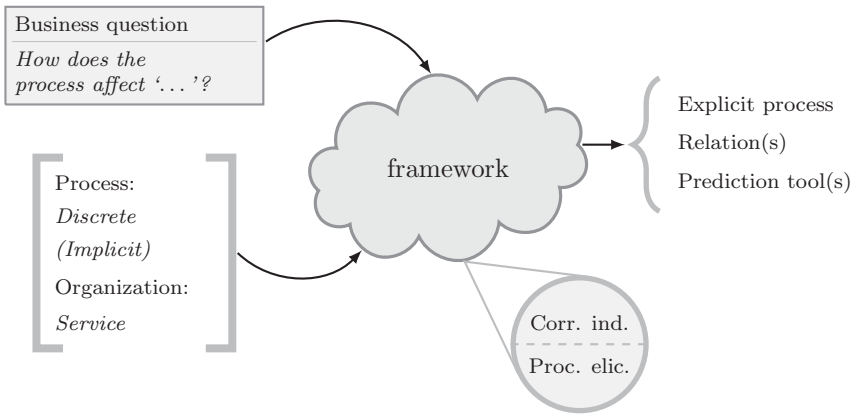


Figure 4.1: Illustration of the research goal

IV. What are, given a method and business question, the process steps we need to take to produce an answer?

The first three questions are about searching existing methods for suitable candidates. This will be covered by literature research, described in chapter 5. The final question is a matter of design, and will be covered in the "Framework"-chapter (chapter 7).

4.2 Approach

There are two deliverables in this project: methods and tools for process elicitation/correlation induction and a set of steps, combining the two, that takes a business question as input and outputs a suitable answer. The first deliverable is the product of literature research amongst candidate methods, tools and techniques. The literature research will start with establishing criteria that can be used to filter out the methods that are most suitable for our purposes. The intermediate product of the literature research is a set of methods. Since we want to use these methods, the next step, in the framework chapter, is to find tools that implement these methods. If no such tools exist, we will implement them.

In the framework chapter, we subsequently derive the processing steps (the second deliverable) from the methods. This is done by looking at the dependencies of methods and which constraints they enforce on how methods are combined. The evaluation chapter describes how this set of processing steps (the 'answering a business question'-process) is validated for the business problem at the hospital. Finally, the conclusion describes the insights from the evaluation and gives answers to the research questions.

Chapter 5

Literature Research

This chapter provides theoretical foundations for the research that is described in this paper. The work described in this chapter is aimed at answering the first three research questions: *What methods exist for finding a process graph?*, *What methods exist for finding the most likely subsequent path through a process graph, given a set of previously traversed nodes?*, *What methods exist for finding path-variable correlations?*

These three questions are answered through a systematic review of existing literature. The chapter is split in two major parts: first, a review of correlation induction literature and second, a review of process elicitation literature. The product (milestone) for this chapter is a set of existing tools and techniques that are suitable for solving the problem class described in the previous chapter, ranked according to the performance measures as described in chapter 3.

5.1 Correlation induction

Figure 5.1 shows the basic concept of correlation induction. Some data is processed to produce a *model*: a set of rules that describe the data that was processed, in other words an abstract representation of the processed data. We use the term *processor* to describe the class of techniques that is able to extract a model from data.

It is from the model that the business question can be answered. For example, if we found a relation such that if a patient receives "cast" as treatment for the diagnosis "broken leg", treatment will be finished within 45 days in 90% of cases and will be DOT product number XX in 80% of cases. It is also possible that the correlations remain implicit, or not human-readable. In this case, a set of performed activities could be input for the set of implicit rules in the model, and the output would be the subsequent path through the process graph (see chapter 3).

The next sections discuss the different types of technique that exist for extracting a model from a provided data set. These methods are all *algorithmic*, which means they use a mathematic procedure.

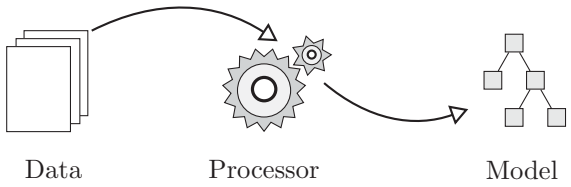


Figure 5.1: Correlation induction: Basic procedure

5.1.1 Data mining

The central term in the goal of this research is finding relations, either those between process steps, those between process step and variable, or amongst process variables. These relations are to be found using historical data. Historical data for the hospital case is data that shows which outcomes are associated with which process steps and the duration of these processes.

A term that is related to correlation is prediction: given variables A and B, we can predict that if variable A has value x, B will have value $x+1$. Prediction is about finding a specific set of relations: it finds relations that occur over time.

Hastie *et al.* [26] state that prediction is part of the field of statistics, or statistical learning. The paper compares three fields: prediction, inference and data mining. What these three methods have in common is that they research how a conclusion can be drawn from a data set. This is also what the prediction system has to do.

Inference is defined in [26] as a "rational degree of belief on non-conclusive evidence". Data mining is different from inference in the sense that inference –more generally classical statistics– uses primary data analysis and data mining uses secondary data analysis. What this means is that classical statistical methods form a hypothesis and check this hypothesis against the data, and that data mining forms a hypothesis by induction on the available data [25].

Hand defines data mining as "the process of secondary analysis of large databases aimed at finding unsuspected relationships which are of interest or value to the database owners" [25]. For this research, the relationships of interest are between process steps and process outcome/duration, or the relation between the path through the process graph traversed up til now and the subsequent future path.

Data mining techniques fall into three more or less distinct categories: *classifiers*, *association rules* and *clustering algorithms* [36].

Tan *et al.*[36] describe these three techniques as mentioned in the list below.

- *Classification* is the task of assigning objects to predefined categories, or classes. Well known classifaction techniques (classifiers) are tree and (Naive) Bayesian classifiers. Many classification techniques require to be trained to work properly.

- *Association analysis* is used to find interesting relationships hidden in large data sets. The analysis results in one or more relationships, called association rules. An assocation rule is found if items in data set frequently occur together.

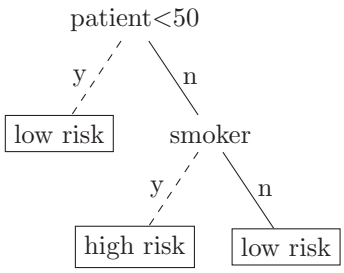


Figure 5.2: An example decision tree

- *Cluster analysis* is used to group data together, so that the data in a cluster is more or less uniform as well as different to the information in other groups. Cluster analysis can be seen as a classification technique, but the classes are not predefined: the found clusters have to be labeled after they have been identified.

Based on the definitions above, we will investigate classification and association analysis further. We will not discuss clustering in detail, as this form of analysis does not use predefined classes, which means it is not suitable to solving the problem presented in this research. Clustering algorithms adapted to classify similar cases will be discussed in the section on classification.

5.1.2 Classification

Han *et al.* [23] describe the following set of classifiers: Decision tree, Bayesian, Rule-based, Neural network, Support Vector Machines and Nearest neighbor classifiers.

These classifiers are discussed in this section. For each classifier, we will give an introduction of the algorithm and how it could be used as a tool for correlation induction. Also, we will present the most suitable implementation or variant of the classifier (if any) and will rank it according to the performance measures presented in the "Concepts" chapter. In addition to this list of classifiers, we will discuss four techniques for augmenting the performance of classifiers: *bagging*[10], *boosting*[18] and *Random Forests*[11].

5.1.2.1 Decision Trees

Decision tree induction is a common technique for classifying data. A decision tree is a representation of a set of classification rules. These rules are induced from the data, either from a sub set or the original set. This induction is called *training*. [32].

Figure 5.2 shows an example decision tree. In the tree, there are two choices: *age* & *smoking* and two classes: *high risk* and *low risk*, denoted by a rectangle. Every path from the root of the tree to an edge is a classification rule, for example: $age < 50 = yes, smoker = yes : high\ risk$.

The set of classification rules (the tree) that is produced by a decision tree algorithm is also referred to as a model in this research, as it is a decision model for deciding the class based on a set of attributes.

Variants and implementations Algorithms for tree induction include Hunt's algorithm, the basis for several well-known tree induction algorithms, such as CART (Classification and Regression Tree), ID3 and C4.5 [36]. Hunt's algorithm recursively grows a decision tree, starting at its leaves. The steps in the algorithm are described below.

- Step 1 is to check if all records belong to the same class a. If so, create a leaf node with label a.
- If the records do not belong to the same class, select an attribute to split the records. Create child nodes and distribute the records over these nodes, based on their value for the selected attribute.

Note that Hunt's algorithm (in this definition) does not specify how an attribute is to be selected. The algorithms that are based on Hunt's all have a different way of attribute selection.

For CART, attribute selection is done with the Gini index. This index is a measure for the inpurity of a node. The goodness of a split with a certain attribute is the consequence of how pure it makes the nodes. The attribute that leads to the lowest index score (lowest Gini index) is chosen to split the records [36].

In ID3 and its successor C4.5, the attribute selection condition is the gain of the attribute [33]. Gain is the difference in information, and is maximal if the weighted average impurity of the child nodes is minimal. This approach differs from Gini in that it takes the information of the parent nodes into account for its measure of the information of the child nodes. The parent information measure is called entropy, which is maximal if "chunks" that the nodes splits the data into are of equal size. The gain measure searches for splits that have minimal entropy, resulting in maximum gain.

Performance The accuracy (percentage of errors) of a decision tree obviously depends on the algorithm used, which in turn depends (amongst other things) on the use of impurity measure. The properties of the data also influence the accuracy of a decision tree, as noisy data sets influence the accuracy of decision trees negatively.

A common problem with decision tree induction algorithms is *overfitting*. Overfitting means that the tree has branches that are the result of anomalous records. These are included if the set on which the tree is trained (alternatively: from which it is algorithmically induced) is (too) large, as this results in "noisy" data to be included as a rule in the tree. These noisy cases may only represent a fraction of a percent of the total data, but large training sets have a greater likelihood of including these noisy cases, leading to overfitted trees. The solution that is employed by algorithms such as C4.5 and CART is to prune the tree: remove branches that likely represent overfitting [23].

Due to pruning, a decision tree is likely to exclude some rules that exist in the data, as these are not common enough in the data to not be pruned. In other words, it is likely that some rules are considered noise. This means that the decision tree approach will likely be incomplete. Also, as pruning is limited, a noisy data set will hinder the accuracy of a decision tree classifier.

5.1.2.2 Bayesian classifiers

A different approach to classification altogether is the Bayesian approach. Tree classifiers are induced from a set of records, Bayesian classifiers use only a classification of a subset of records and rely on a measure for the likelyhood of a record t belonging to a class C_i. The measure is given by the formula known as Bayes' rule:

$$P(C_i|t) = \frac{p(t|C_i)P(C_i)}{p(t)}$$

The class with the maximum value for $P(C_i)$ is the class that is assigned to record t. $p(t|C_i)$ is known as the class likelihood [7] and is the probability that a record belonging to C_i has the attributes of record t. $P(C_i)$, prior probability, is the probability that a record belongs to class C_i, regardless of the attributes of that record. $P(t)$, the evidence, is the probability that a record has the attributes of record t. The result of the calculation, $P(C_i|t)$ is known as the posterior probability. The class that has the greatest posterior probability for a record is the class label applied to that record.

Variants and implementations The problem with Bayes' rule is that the class likelihood is sometimes difficult to calculate. Some sort of initial classification is needed to decide which atribute-value pairings belong to which class. To do this requires information on how the values for different attributes are related. For example, if a record has attributes-value pairs "shape=round" and "color=green" it is classified as "apple". In this case, the two attributes are dependent. *Naive Bayes* is a variant of Bayes' rule that is easier to use in practice. It is based on the assumption that the attributes of a records in the set are independent. Under this assumption, class likelihood is given by:

$$p(\{v_1, \ldots, v_i\}|C) = \prod_{j=1}^{i} p(v_j|C)$$

Here, $\{v_1, \ldots, v_i\}$ denote the values of the attributes for some record for which the class likelihood for class C is calculated.

In practice, the naive Bayesian classifier is the more obvious choice over a non-naive Bayesian classifier, as there is often no information on how to label certain variable pairings, and only new records that have a pre-existing pairing can be classified. For example, if the pairing green-round-sweet is labelled as an "apple" and occurs in 1000 records and the pairing yellow-elliptical-sour is labelled as "lemon", Naive Bayesian would label a new record with pairing red-round-sweet as an apple. Non-naive Bayesian would be unable to classify this new record.

Another implementation of Bayesian classification is the *Bayesian belief network*: a graphical representation of a Bayesian classifier. This type of network is also known as Bayesian network and probabilistic network. Figure 5.3 shows an example Bayesian network.

The probability of the edges is decided as a function of the probability of the node they enter. Here, that means that the probability of "Heart condition" decides the probability of its incoming edges [7]. As the graphical representation roughly matches that of a decision tree, we briefly discuss the difference: A

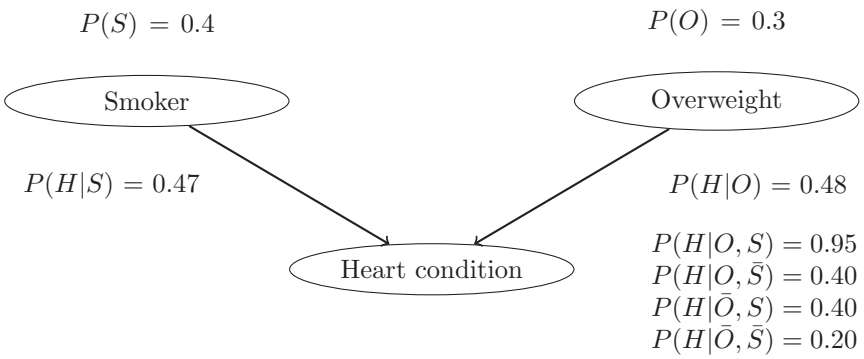


Figure 5.3: An example Bayesian network

Bayesian belief network is an acyclic directed graph, whereas a decision tree is an acyclic connected graph with for each node at most one parent. In a Bayesian network, the route through the graph is decided based on atribute probability, in a decision tree it is an attribute condition.

Performance The assumption of variable independence means that pairings are not taken into account. While this makes the classifier applicable to most data sets (see the constraints on non-naive Bayesian above), it also means that variables that should be paired are in fact not. This means that in a data set of 1000 lemons and 5 apples, something round, green and sweet will not be classified as an apple. More generally, this means that the naive Bayesian approach is possibly inaccurate for datasets that consist of dependent variables, which is a likely scenario.

As Naive Bayesian does not produce rules, but instead uses Bayes' rule to label records, the completeness measure is not applicable to this classifier.

5.1.2.3 Rule-based classifiers

Although rule-based classifiers are part of the classifier category in the definition of [23], they are more strongly related to association analysis, since they find relationships between variables, not classes based on variables. The rule-based classifiers are discussed in detail in Section 5.1.3.

5.1.2.4 Neural networks

An example neural network, or multi-layered perceptron, is shown in figure 5.4. It consists of multiple elements that mimic biological neurons. These elements are called perceptrons, every node in figure 5.4 represents such a perceptron.

A perceptron (as shown in figure 5.5) functions as follows:

1. The perceptron receives input: a set of values (a vector)

2. The input is combined with a pre-existing set of weights for each input value. These weights are the result of training the perceptron.

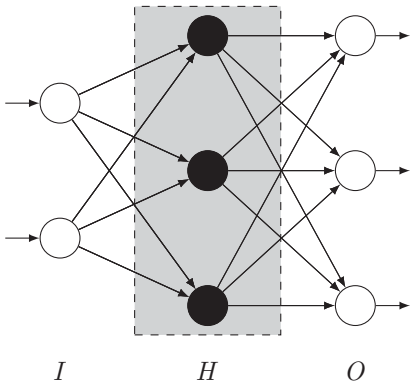


Figure 5.4: A neural network with one hidden layer

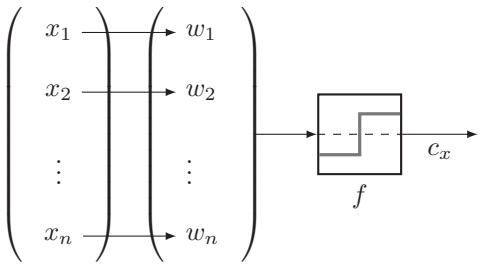


Figure 5.5: Diagram of a single perceptron

3. The combined input and weight are passed through a threshold function f. If the input/weight are above a certain value (again decided by training the perceptron), the input vector is classified as c_2. Otherwise, its class is c_1. The output could serve as input for another perceptron.

Multiple perceptrons tied together –which is the case in a neural network– mimic the function of a (human) brain. The idea is that a single perceptron (or neuron) is a simple processing unit, but tied together through a large number of *synapses* (in human brains neurons are connected to 10^4 other neurons) and operating in parallel they provide significant computational power [7].

In neural networks, perceptrons are placed into different layers: one for input (marked I in figure 5.4), one or more hidden (marked H) and one output layer (O). The input layer takes the attributes of the instance to be classified as input, weighs them and gives a class as output. The hidden layer(s) take classes from their input perceptrons as input, weighs them and give a class as output to (eventually) the output later, which does the same.

The hidden layer is an extra processing step, that is used to increase the classification performance of the neural network, especially when analyzing non-linear functions (where the input and output are not linearly related) or continuous variables.

As mentioned before, individual perceptrons are trained in order to set the

weights for their inputs and threshold. This is done by "feeding" the perceptron a set of inputs and the class they belong to. The perceptron starts with random weights, then sets its weights and threshold based on an underlying algorithm, that involves adjusting the weights and threshold to minimize the classification error.

Since not only the individual perceptrons matter, but also the network they are part of, the network as a whole is trained as well. The algorithm that performs this training is known as backpropagation and is based on the concept that errors in the output layer perceptrons are the result of errors in the hidden layer perceptrons and hidden layer errors are caused by errors in the input layer. Effectively, this leads to a weighing of perceptrons as more or less important by their subsequent perceptrons [7][23].

Variants and implementations There are two points on which the neural network can be tuned to match the situation in which it is used:

- *The (size of) the training set.* This influences accuracy, but a too large training set leads to overtraining. As mentioned before, crossvalidation is a tool to establish the maximum size for the training set without overtraining effects.

- *The number of hidden layers.* More hidden layers makes the network a better representation of non-linear relations, but also cause problems similar to overtraining. If the number of hidden perceptrons increases, the mean square error of the multi-layered perceptron increases drastically [7].

Performance Having a large training set will result in the weights of the (multi-layered) perceptron to match the training set as best as possible. The result is that the neural network may perform well on the training set, but perform poorly on a real set, either due to noise in the training set or because the instances in the training set do not reflect the actual population accurately. The problem is similar to that experienced in decision tree induction, there it is called *overfitting*, for neural networks it's called *overtraining*. The solution is also similar: reduce the size of the training set. The ideal training set is found through cross validation with "new" sets of instances (that have not been used in training).

A different limitation altogether is the hardware on which the neural net operates. A neural network consists of parallel executing perceptrons, maximizing the speed of the network relies on the underlying hardware being able to cope with this parallelism.

In practical use, such as in a business situation, the people who are investigating the relation between variables that relate to their business processes will perhaps want to know the rationale between the relations an algorithm finds. Understandability of a neural network is limited, due to the hidden layers, that are basically a black box: we know the input and output, but we don't know what happens in between. Also, the weights of the perceptrons or what they're based on is a "mystery". Although this is not actually a problem if the network performs acceptably, some users will find its conclusions unreliable, if they don't understand what's going on "under the bonnet".

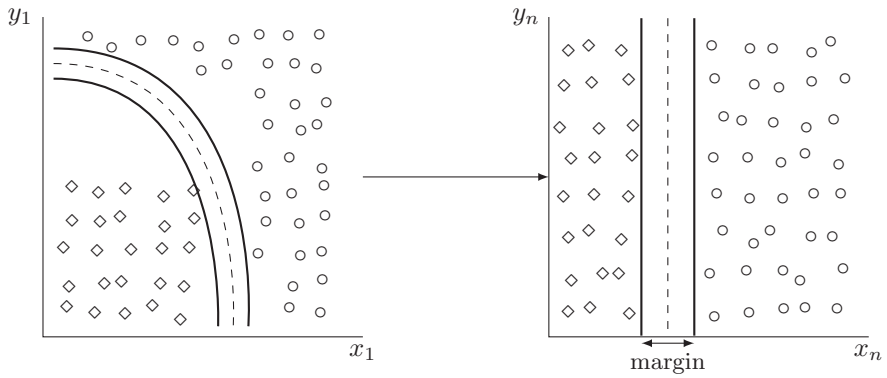


Figure 5.6: Support vector machine: hyperplane and margins

5.1.2.5 Support Vector Machines

A support vector machine or support vector network is a method for solving two-group (binary) classification problems [13] [30].

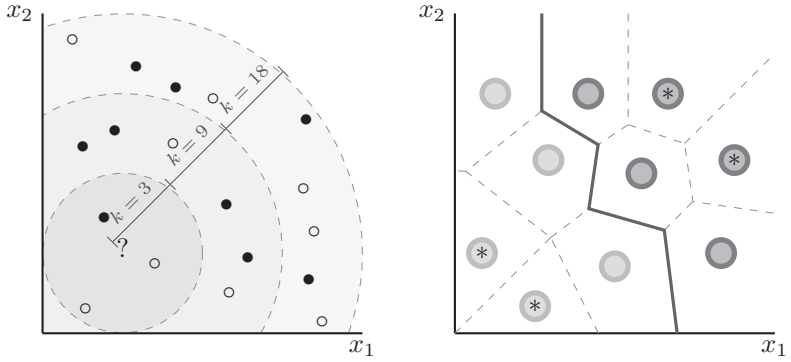
Figure 5.6 shows two graphs of nodes. The left graph is in 2-dimensional space, the right represents the same group of nodes, but mapped to some n-dimensional space. The n-dimensional space is chosen such that the two classes are linearly separable in that space. In the figure, this is the case in the right hand graph.

Both graphs have a line which optimally separates the two classes $\diamond$ and $\circ$, called the hyperplane. The thick lines represent the margin of the hyperplane, or how distant the hyperplane is from the nearest node. The standard SVM algorithm [13] picks the hyperplane that maximizes the margin. The terms *hyperplane* and *support vector machine* are interchangeable [7].

Once the hyperplane is algorithmically derived, it is used to classify new instances (nodes): if a node has a value $n(x) > H$, choose c_1; otherwise, choose c_2, with H a hyperplane, $n(x)$ the value for a node attribute and c_1 and c_2 as classes.

Performance Support Vector Machines perform comparable to other classification and regression techniques [30]. However, they are outperformed by for example random forests and bagging. This shows that they are a candidate procedure for relation induction, but are likely less reliable than some other techniques.

A further limitation still is that support vector machines split nodes into only two classes. This means that their applicability in multiple-class problems is debatable, as it will require some alterations to the algorithm. We have not been able to identify research that deals with this limitation, therefore we conclude that SVM's are only to be considered as a relation induction algorithm for binary problems.



(a) k Nearest Neighbors **(b)** Condensed Nearest Neighbors

Figure 5.7: Nearest Neighbor Algorithms

5.1.2.6 Nearest neighbor classifiers

Nearest-neighbor classifiers use the distance to other instances in a graph to decide the class of a new instance. Since they do not use parameters of the instances directly, they are also known as non-parametric classifiers [7].

This sets this group of classification algorithms apart from those discussed before: they are all trained (or work on) the attributes or parameters of instances and classify new instances accordingly.

Another term that is used synonymous with "non-parametric classifier" is *clustering algorithm*, or an algorithm that groups instances based on proximity [36].

Variants and implementations We will discuss two algorithms that fall into this category: *k-Nearest Neigbors* (kNN), *Condensed Nearest Neighbor*. An example for both algorithms is shown in figure 5.7. The basic principle of the kNN-classifier [7] is shown in figure 5.7a. The question mark represents a new instance. If we take $k = 3$, it is classified as ∘, as there are two of that class, versus just one of the other class. If we take $k = 9$, the choice is •, as there are five •'s to four ∘'s. For $k = 18$, it is somewhat more difficult, as there are an equal amount of instances for either class. The choice of class is then made based on chance (flipping a coin) or by attaching weights to each neighboring instance. The latter choice is, in other words, dependent on the implementation.

A slightly different nearest-neighbor algorithm at work is shown in figure 5.7b. The thick line represents the border between the two classes (represented in light and dark gray). A new instance that lies within the area to one side of the discriminant (shown as a thick gray line) is classified as the class of the instances that border the discrimant on that side. If the discriminant was properly trained, the new instance will border only instances that border the discriminant, but not the discriminant itself.

The instances marked with an asterisk can be removed without effecting the discrimant and therefore increasing training error. Training the discriminant is

done by for each training set instance, seeing if it can be correctly identified using the discriminant. If not, the instance is added to the set of discriminant instances and the discriminant is changed accordingly. In fact, this is a variant of kNN –with $k = 1$– as instances are assigned the class of its direct neighbor [7]. For example, the instances marked with an asterisk are classified as their neighbor, which borders the discriminant. Cover and Hart have shown that the error rate of an kNN classification is at maximum as low as it is for $k = 1$ [14]. This means that Condensed Nearest Neighbor will have the lowest error rate achievable for a kNN-classifier.

Performance The kNN-algorithm is sensitive to noise: given a data set where the n classes are distributed (nearly) uniformal, an instance has near equal chance of assigning any class c_n. The error rate will then approach $\frac{1}{n}$, in other words, the performance will not be better than for an arbitrary (random) classifier. This is an extreme situation, but needs to be taken into account when choosing a classifier.

5.1.2.7 Classifier performance augmentation

The performance of a classifier is defined as inversely related to the number of classification errors it produces. The performance of a solution to a classification problem does not merely depend on the fit of the classifier to the problem, or more precisely, we can influence the performance not only by choosing the right classifier, there are also different ways of augmenting the performance.

This section discusses such augmentation techniques, namely *bagging*[10] and *boosting*[18]. The techniques both use the principle of taking multiple classifiers or classifier instances and having them vote on the class of an instance. The majority decides the class.

Bagging –an acronym for *b*ootstrap *agg*regating– uses an unweighted vote. That means that every classifier's vote contributes equally to the choice of class for the instance. Bagging can be used on a multitude of classifiers, but achieves the best performance when used with a classifier that itself performs well, as tested with the C4.5/J48 tree classifier mentioned before [10].

The bagging technique takes multiple classifiers and trains them on a sub set of the training data. It then aggregates the results, thereby pulling itself up "by the bootstraps". More formally, the procedure starts with a learning set $\mathcal{L}$, which is split into k subsets $\mathcal{L}_k$. For each of these subsets, a predictor $\phi(x, \mathcal{L}_k)$ is trained, where the class y is predicted by input x (Predictor is used synonymously with classifier). What this means is that each predictor votes on the class for y, the class with the most votes is chosen as the label for y.

The bagging procedure performs best with unstable results for the classifiers, because this makes it possible to filter out bad results by voting. Breiman refers to his technique as "making a silk purse out of a sow's ear" to describe that it can take a reasonably performant classifier and turn it into one that performs remarkably well. A downside of the technique that it obscures its inner workings, as the underlying classifiers that produce the result are hidden from view.

Boosting is a technique similar to bagging, in the sense that it uses a vote on the classification of an instance y. However, where it differs is that in boosting,

different weights are assigned to classifiers. In essence, boosting algorithms give more weight to the training instances that are more difficult to train.

A well known boosting algorithm called AdaBoost was created by Freund and Schapire, as described in [17] [18]. This takes a weak learning classifier – or one that does not perform well on a training set– and "boosts" its results. The algorithm, more specifically the variant for two or more possible classes Adaboost.M2 [18], takes the steps below to reach a hypothesis on the class y belonging to some instance x.

1. Use the classifier to give a vector $(0,1)^k$ of class probabilities for the k classes, by feeding it a distribution of example-incorrect label pairs. This distribution is called a mislabel distribution, which Freund and Schapire define as a distribution D_t over the set B of pairs (i, y) where i is a training example and y a class incorrectly associated with that example (formally, $B = \{(i, y) : i \in \{1 \dots m\}, y \neq y_i\}$, for a total number of instances m). In D_t weights are assigned to each mislabeled pair, where higher weights stand for more difficult examples.

2. The classifier produces a vector $X \times Y \rightarrow [0, 1]$ for the pairings, where 1 stands for high probability, 0 for low. What this means is that if the classifier gives a value (near) one for a pairing of instance i and (correct) class label y_i and a value near zero for the pairing of that instance and mislabel y, it concludes that the (correct) classification y_i for that instance is plausible. If the value for the (i, y_i) is (near) zero and vice versa for the incorrect pairing, according to the weak classifier, the instance should be labelled with the incorrect class label y. Values around 0.5 indicate that the classifier evaluates the incorrect and correct pairings as equally plausible.

3. The next step is to determine the *pseudo-loss* of the hypothesis that was produced by the classifier, a measure proposed by Freund & Schapire. The thought behind the pseudo-loss is that it emphasizes important misclassifications, by using the (weighted) mislabel distribution D_t. The lower the pseudo-loss is, the better the hypothesis addresses the instances that are difficult to classify.

4. The weight for the instances that were classified in the hypothesis is updated, which in turn is used to compute distribution D_{t+1} from D_t. Misclassified instances get a higher weight than correctly classified instances, therefore realizing the principle of the boosting algorithm: the predictor that is good at classifying the difficult cases has a stronger weighing vote.

The output of the algorithm (that is, instance-class label pairings) is computed by finding the y which has the maximum weighted average value amongs the T hypotheses for an instance x. In this computation, hypotheses with high pseudo-loss are given little weight in determining the final hypothesis.

Performance The boosting algorithm AdaBoost needs a weak classifier that performs (slightly) better than random [18]. In terms of the algorithm, a random classifier would give every pairing (i, y) the same likelihood, i.e. 0.5.

The "boost" achieved by this algorithm has been experimentally [18] shown to be better than bagging in most cases when combined with a weak classifier. Using a strong classifier (one that itself has good performance) the results of bagging and boosting are comparable. Furthermore, using the boosting algorithm makes sense in combination with a strong classifier, as it still increases performance in most cases, although it hurts performance in a few classification problems.

A simpler variant of the AdaBoost algorithm called Adaboost.M1 (which differs from M2 in its error measure, it does not use pseudo-loss), shows more consistent but averagely lower performance than M2. The combination AdaBoost-weak classifier has comparable performance to strong classifiers such as C4.5 [33].

5.1.2.8 Random Forests

A classification technique that shares elements from both decision tree induction and the augmentation techniques mentioned in the previous section is called *Random Forests* [11]. The "forest" refers to a group of decision trees, the "random" part refers to how those decision trees are induced (or grown).

As mentioned in the section on decision trees, decision tree induction algorithms use different measures to find the features to split the data set on such as Gini-index, Entropy and Split Info. The decision trees that make up the forest in the random forest technique take a *random* set of features to split the data on. The size of this set (with upper bound the total number of features) is one of the two parameters of the random forest, the other being the number of trees. The algorithm is described more formally in the next paragraph.

Algorithm For K trees, and $k \in K$, generate a random vector Θ_k for the kth tree such that Θ_k uses the same distribution and is independent of $\Theta_1, \ldots, \Theta_{k-1}$. This vector (of features) is applied to the training set to grow a decision tree $h(x, \Theta_k)$ that takes some vector x as input and produces a class label for that vector.

The classifiers $h(x, \Theta_1), \ldots, h(x, \Theta_K)$, take a vote on the class for input vector x. The most popular vote decides the class for x. Figure 5.8 shows this process graphically.

Performance According to the developer of the random forest algorithm (the same as for bagging and arcing), Leo Breiman, an advantage of the algorithm is how well it deals with noise [11]. The algorithm shares parts of its procedure with the bootstrap aggregating, namely having multiple (weak) classifiers vote on the class label for some instance.

Where they differ is that bagging uses the same classifier every time, but trains it on different sub sets of the data [10] whereas the random forest uses the same training data every time, but takes a different feature set to train the tree with [11]. The classifiers for the bagging procedure will have comparable performance if they have comparable data sets, the trees in the forest might all perform very differently, due to the different choice of splitting features. This means that performance variances amongst classifiers will occur in both procedures, but what's causing them is different for both.

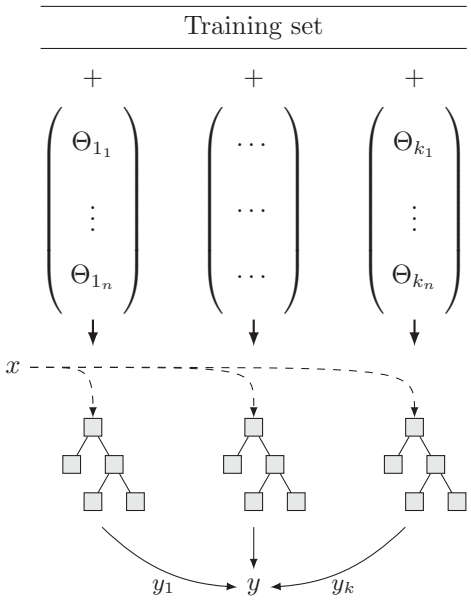


Figure 5.8: Random Forest classification

Still, the similarities of both procedures suggest that the random forest will have comparable performance to a bagging classifier. According to Breiman, the performance in terms of accuracy is comparable to AdaBoost (the author does not mention which variant), it is not sensitive to noise, it is faster than bagging/boosting and is easily parallelized.

These claims are of course possibly somewhat biased, but experiments show that its performance is better in eleven out of nineteen test sets [11]. This result is corroborated by [9] [19], those authors also found that random forests regularly outperform AdaBoost and that both algorithms outperform all the other techniques they tried. This clearly shows that the random forest algorithm is a high performing algorithm in terms of accuracy. Gislason *et al.* [19] and Breiman [11] also found the algorithm to be faster than bagging and boosting.

5.1.2.9 Summary of classification techniques

Table 5.1 summarizes the findings on classification techniques in this section, stating their relative performance in terms of accuracy, well-known algorithmic implementations (if the technique itself is not an implemented algorithm) and their limitations. Note that where understandability is mentioned as a limitation, we mean that the technique does not produce a model that is readable to humans, or does not output a model at all.

The accuracy in the table refers to the average number of misclassifications by the technique, compared to the other techniques in the table. These are all the best-case scenario, or a situation for which the algorithm is suitable. For instance, a Naive Bayesian algorithm performs best where the data features are independent, as it assumes independency. A ++-sign means that the technique performs amongst the best of those compared, a + denotes it performs aver-

Technique	*Acc.*	*Algorithms*	*Limitations*
Decision trees	+−	Classification and Regression Trees (CART), C4.5	Overfitting, sensitive to noise.
Bayes	+−	Naive Bayesian classification, Bayesian Belief Networks	Naive Bayes assumes independency of data set features. If this is not the case, performance is harmed.
Neural networks	+[5][15]	Backpropagation, Feed Forward	Understandability
Support Vector Machines	+− [30]	Support Vector Networks	Only usable for binary classification problems
Nearest Neighbor classifiers	+−	k-Nearest Neighbors, Condensed Nearest Neighbors	Understandability
Bagging	+	Bagging + strong classifier/weak classifier	Bagging works best with unstable underlying classifiers [10]
Boosting	++ [18]	AdaBoost.M1, AdaBoost.M2	AdaBoost.M2 has more unstable results than the M1 variety, but has higher performance.
Random Forests	++[9] [11] [19]	-	Understandability

Table 5.1: Comparison of classification techniques

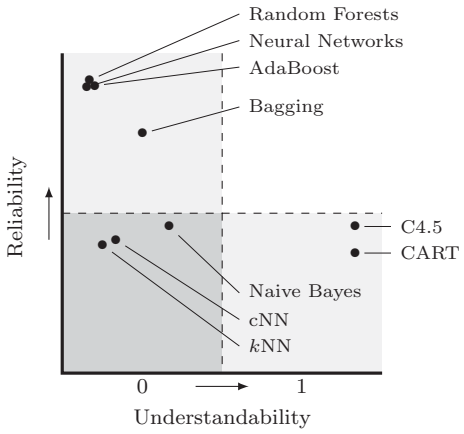


Figure 5.9: Comparison of classifiers

agely, +− denotes sub-average performance. The referred papers have found to the technique to be either performing better or worse than a respectively higher or lower ranked technique. For example, [15] shows neural nets to be performing better than tree induction, whereas [5] found them to perform worse than boosting.

From the previous sections, as summarized in table 5.1, we conclude that *neural networks*, *boosting* and *random forests* are the classification techniques that are the most promising, as they are most accurate. However, if we look at figure 5.9, we see that none of the best performing classifiers actually produces a human understandable model. If human understandability is required for the model, the tree classifiers are the best choice.

5.1.3 Association analysis

Where classification is used to find the correct class label that belongs to an instance, either based on its features or how they're grouped, association is about finding frequent patterns, correlations and causal structures that occur in data [29].

A pattern is a set of features that is often found together. Early applications of association analysis– or association rule mining as it is also known– include finding patterns in the "basket" of supermarket shoppers [2].

5.1.3.1 Some definitions

Agrawal *et al.* give the subsequent formal definition of an assocation rule [2]:

Let $\mathcal{I} = I_1, I_2, ..., I_m$ for some m: a set of binary attributes or *items*. T denotes the set of transactions. Given a vector of attributes $t \in T$, where $t[n] = 1$ stands for a positive value for the nth attribute, t satisfies some $X \subseteq \mathcal{I}$ iff $\forall k\{t[k] = 1 | I_k \in X\}$. An application rule is of the form $X \Rightarrow I_j$, where I_j is a single item. It is a rule if at least $c\%$ of the items in T that satisfy $X \subseteq \mathcal{I}$ also satisfy I_j. This percentage c is called confidence in a rule $X \Rightarrow I_j$.

Transaction	*Items*
1.	{Coffee, bread, milk, cheese }
2.	{Bread, bananas, cheese}
3.	{Cheese, coffee, milk, apples}

Table 5.2: Example transactions

Confidence-type measures are what decide what a association rule is and what is not. If the proposed rule scores above some threshold, it's a rule, otherwise, it's not. Below are a few measures of reliability of a rule $X \Rightarrow I_j$:

- *Confidence (c)*: The percentage of transactions that satisfy I_j as well as X.
- *Support count* (σ): The total number of transactions that satisfy X, denoted $\sigma(X) = |\{t[k] = 1 | I_k \in X\}|$ [36].
- *Support (s)*: The fraction of items that satisfy X, or the ratio of support count for X to the total size of the transaction set, denoted $s(X) = \frac{\sigma(X)}{|T|}$.

Table 5.2 shows a set of transactions T. The support count for the itemset X ={Cheese, Bread} is 2, making its support $\sigma(X) = \frac{2}{3}$. The confidence of the rule {Coffee, Milk}⇒{Bread}= $\frac{1}{2}$, or fifty percent. The support for that rule is equal to the support for {Coffee, Milk, Bread} divided by $|T|$, or $\frac{1}{3}$ for this example.

An itemset that has more than some threshold value for support (called *minsup*), it is called a *frequent* itemset. Similarly, if an association $X \rightarrow I_j$ has confidence higher than *minconf*, it is a *strong* rule. The goal of association mining is to find all the rules that have confidence greater than *minconf* and support greater than *minsup*. Tan *et al.* give the maximum number of rules R to be found in a set of transactions T:

$$R = 3^{|T|} - 2^{|T|+1} + 1.$$

For the example set of items in table 5.2, this means there are a total of 698 possible rules. For a set of 10 items, there are already 57002 rules. For 1000 items, the amount of possible rules exceeds $1,3 * 10^{477}$. Considering that association rule mining started out as a technique for large databases with large itemsets [2], there must be a smarter way to deal with rule mining than just finding every possible rule and seeing if it has high enough confidence and support. This is where algorithms come into play, such as the Apriori algorithm and some variants approach that will be discussed in the next few sections. To start, we will present some steps that are part of a generic procedure for association rule mining.

5.1.3.2 General association rule mining procedure

Tan *et al.* claim that no more than twenty per cent of possible rules will have confidence greater than the confidence threshold *minconf* [36]. This means that finding for a data set with one thousand items there are approximately

$3,3 * 10^{476}$ meaningful rules (strong rules). Again, this set is to large to traverse economically.

Luckily, there are some "tricks" that can be used to reduce the number of rules that has to be checked for strength. The confidence measure of a rule $\{X_1, X_2, \ldots, X_n\} \Rightarrow \{Y_1, Y_2, \ldots, Y_n\}$ is equal to the support count for the union of the antecedent set (labeled X) and the consequent set (labeled Y) divided by the support count for the consequent set[2]:

$$c(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(Y)}.$$

This is a useful piece of information, as we can optimize the search for frequent itemsets (itemsets with support $>$ *minsup*): If a set $X = \{X_1, \ldots, X_n\}$ has support below threshold, all itemsets $\{Y | X \subset Y\}$ will be infrequent. What the definition of confidence also means is that all rules where the antecedent is infrequent will have a confidence $\leq$ *minsup*.

For instance, with the transaction set in table 5.2 and given *minsup* (0.4), the set $X = \{Bread, bananas\}$ is infrequent (with $s(X) = \frac{1}{3}$). Therefore, any rule with X as antecedent has confidence $\leq$ *minsup*: $X \Rightarrow \{Bread\}$ has confidence $\frac{1}{3}$, $X \Rightarrow \{Milk, cheese\}$ has confidence $\frac{0}{3} = 0$.

This leads to the general procedure for association rule mining, as shown in algorithm 5.1. The procedure entails finding the subsets that have enough support and ignoring those that don't. After performing the procedure, only frequent subsets remain. If a set $\{X, Y, Z\}$ is frequent (has support $>$ *minsup*), then a rule $\{X \Rightarrow Y, Z\}$, $\{X, Y \Rightarrow Z\}$ or any other variant will have support above the threshold: For a three item set to be considered, its two and one item subsets will have to have support $>$ *minsup*. A superset $\{X\}$ has, by definition, support $S(\{X\}) \leq S(\{X' | X' \subset X\})$. Therefore:

$$(s(X') \geq minsup) \rightarrow (\frac{\{X' | X' \subset X\}}{X} \geq minsup)$$

The astute reader will have noticed that the fraction $\frac{X'}{X}$ does not, in fact, denote support, but confidence. If *minconf* $\leq$ *minsup*, this means that all the rules that can be deduced from the procedure's output have at least confidence *minconf*. If *minconf* $>$ *minsup*, the amount of candidate rules is significantly reduced to make it feasible to calculate the confidence for every rule individually. Note that the support for a rule $X, Y \Rightarrow Z$ is trivial, as the set X, Y, Z has – by procedure 5.1 – support $>$ *minsup*.

This general procedure is also known as the *Apriori*-algorithm, as defined by Agrawal *et al.* [4]. The algorithm is not very efficient, as it, amongst other things, requires multiple passes over the database and sometimes requires more computations to be performed on the procedure's output[35]. Note the difference with classification algorithms: for classifiers, the performance in terms of error rates is what sets them apart, whereas association miners will all find all rules with sufficient support and confidence, but arrive at that result more or less efficiently.

Numeral algorithms have been developed that are more efficient, such as *partitioning* [35], *hashing* [31], *sampling* [3] and *FP-growth*[24][45]. All these algorithms are variants on *apriori* and use the fact that subsets of frequent itemsets are also frequent and its corollary: if a subset is infrequent, so will its supersets be.

```
for each {i|1 ≤ i < |I|} do
  someFrequent = true
  while someFrequent do
    for each {I_i = {I_1, ..., I_n}|I_i ⊆ I ∧ |I_i| = i} do
      if s(I_i) > minsup then
        someFrequent = true
      else
        someFrequent = false
        for each {t|t ∈ T ∧ I_i ⊆ t} do
          Remove t
        end for
      end if
    end for
  end while
end for
```

Algorithm 5.1: General procedure for association rule mining

5.1.3.3 Partitioning

The partition-algorithm is different from Apriori in that it has fewer passes over the data set: two for partition versus up to $|\mathcal{I}|$ (where $\mathcal{I}$ denotes the set of items in the database) for Apriori. As the algorithm's name suggests, this efficiency gain is achieved by partitioning the data. A partition is a subset P of the set of transactions $\mathcal{T}$ of size $\frac{|\mathcal{T}|}{n}$, where n denotes the number of partitions.

A partition does not overlap other partitions, to put it differently, transactions $t \in \mathcal{T}$ that appear in partition P_i do not appear in any partition $\{P_x|P_x \in \{P_0, \ldots, P_{n-1}\} - P_i\}$. The number of partitions is chosen so that it matches the amount of memory available in the device that is running the database, so that a partition "fits" into memory [35].

The algorithm consists of three steps, of which the latter two are different from Apriori:

1. Find the frequent 1-itemsets by making a pass over the database [1].

2. Generate the frequent itemsets for every partition. Here, an itemset is frequent if it has support and confidence higher than respectively *minsup* and *minconf* in the *partition*. This support measure is called *local support*. Parallelly, support in the set of transactions as a whole is called *global support*.The frequent itemsets are called local frequent itemsets and are not necessarily frequent in the set of transactions as a whole – There are so-called false positives. Generating a set of local frequent itemsets does however not result in *false negatives*, or frequent (global) itemsets that were mistaken for infrequent (local) itemsets.

3. Generate all frequent *global* itemsets from the merged set of frequent *local* itemsets. This takes n iterations, where n (again) denotes the number of partitions of $\mathcal{T}$. In every iteration, the support count for every itemset in the partition is calculated. The sum of the support count for an itemset for

[1] Savasere *et al.* do not include this pass in their claim that partition only takes two passes

every partition is the global support count. Dividing the global support count by the cardinality of the transaction set $\mathcal{T}$ gives the global support for an itemset. If this is above the *minsup* threshold, it's a frequent global itemset.

What sets the partition algorithm apart from Apriori is the way it finds the (local and global) frequent itemsets. It starts with making a list of all transactions that contain a frequent item $i \in \mathcal{I}$ for all the frequent 1-itemsets in the partition, denoted $L_1 P_j$. Those lists of transactions are called *tidlists* (transaction-id-lists). The local frequent sets larger than one are calculated by for every $2 < k < |\mathcal{I}|$ creating the possible k-itemsets from combining every two frequent $(k-1)$-itemsets l_1 and l_2. For every created combination X_k, the cardinality the intersection of the tidlists for l_1 and l_2 is divided by the cardinality of the partition to give the support:

$$s(X_k) = \frac{|tid(l_1) \cap tid(l_2)|}{|P_j|}.$$

If $s(X_k) \geq minsup$, X_k is added to the frequent k-itemset:

$$L_k^{P_j} = L_k^{P_j} \cup X_k.$$

The global support count for a k-sized itemset is calculated by counting the number of transactions in the tidlists for every one item set $i \in k$. Formally:

$$\sigma(\{X_1, \ldots, X_k\}) = \sum_{j=0}^{n} |\bigcap_{i=1}^{k} tid_{P_j}(X_i)|.$$

Every X_i denotes a single item, $tid_{P_j}(i \in \mathcal{I})$ is the tidlist for an item i in the jth partition P_j of a total of n partitions. The support for the k-itemset $X = \{X_1, \ldots, X_k\}$ in the set of transactions $\mathcal{T}$ (global support) is given by

$$s(X) = \frac{\sigma(X)}{|\mathcal{T}|}$$

Savasere *et al.* found their algorithm to perform up to four times better than Apriori in terms of processor use and an order of magnitude better for input/output [35].

5.1.3.4 Hashing

The hashing algorithm, or *Direct Hashing and Pruning* (DHP) [31] is – just as the partition-algorithm – an extension of Apriori. It is supposed to be more efficient in generating frequent itemsets and reduces the size of the transaction set $\mathcal{T}$ it has to process. As its name suggest, this algorithm achieves its efficiency gain by using hashing.

The DHP algorithm uses the same principle as Apriori: use the frequent L_i-itemsets to generate the L_{i+1}-itemsets. DHP is designed to significantly reduce the number of candidate frequent itemsets for the smaller itemsets. This is efficient, as the set of candidate frequent items for L_1 and L_2 is likely to be the largest. Going over all the candidates to establish their support is therefore expensive, making reducing the number of candidates a desirable feature.

DHP excludes candidates based on hashing: while scanning the transaction set for the k-sized candidate itemsets, the candidate $(k+1)$ items for every transaction are hashed to a hash table through some function h. The value of the buckets in the hash table represents the number of transactions that satisfy the itemset that maps to the key to that bucket.

In the next pass of $\mathcal{T}$, a bit vector of the buckets in the hash table is used to determine if a transaction needs to be included while counting the support of the $(k+1)$ candidate itemsets.

The effect of excluding candidates and trimming the database is that the size of the candidate 2 item is reduced significantly compared to Apriori (it is an order of magnitude smaller). This leads to the total execution time being significantly lower than Apriori as well. A downside of DHP is that it needs to generate a hash table from the k-sized permutations all transactions for all k that have a frequent itemset. For a large dataset, this is a big computational overhead, meaning it might not outperform Apriori at all. In fact, Park *et al.* have shown that for low *minsup* – leading to a high number of itemsets to consider – the difference between Apriori and hashing is small. Therefore, we can conclude that hashing is an improvement of Apriori for relatively small datasets.

5.1.3.5 Sampling

Sampling is a simplification of the Apriori algorithm, that in fact loses some of its properties: the sampling variant does not satisfy *completeness* [3]. The algorithm takes a sample of the transactions in $\mathcal{T}$. Given an itemset X with some support $s(X)$ and $h \subset \mathcal{T}$ sample transactions , the number of transactions containing X –denoted x– has a binomial distribution over h, with success probability $s(X)$. This x is an estimate of the support for X. The probability that x differs more than α from the actual support for $s(X)$ is given by:

$$P[x > h(s+\alpha)] < e^{\frac{-2\alpha^2 h^2}{h}} = e^{-2\alpha^2 h}$$

What this means is that to make the chance of the support estimate exceeding the actual support by more than $\alpha \approx 0.01$ for $\alpha = 1\%$ the sample size h needed is around 23000. More brevily, to get

$$P[\epsilon > \alpha] \approx 0.01, \alpha = 0.01$$

h needs to be 23000. For $\alpha = 0.001$, h needs to be a factor 100 larger. This can be problematic if the *minsup* is set low, as this likely combines with wanting a low α, such as $0.001 > \alpha > 0.0001$. For large databases and for itemsets that have high support however, *sampling* can be an effective and efficient approach.

5.1.3.6 FP-growth

The final association rule mining-variant we discuss is *frequent pattern* or FP-growth. It differs from the other variants in that it does not generate candidate frequent itemsets [24]. The approach involves growing a frequent pattern or FP-tree, hence the name. Like most of the variants we discussed before, FP-growth aims to alleviate Apriori's main bottleneck: generating candidate-itemsets and testing them.

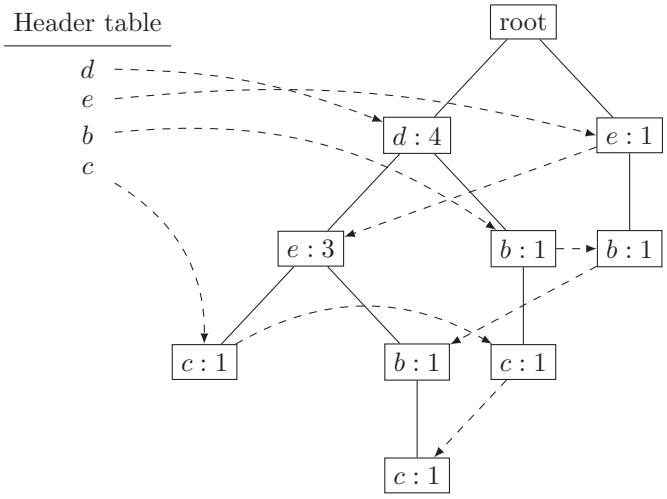


Figure 5.10: Frequent Pattern-tree

TID	*Items*	*Freq. items*
100	$\{b, c, d, e, f\}$	$\{d, e, b, c\}$
200	$\{b, c, d\}$	$\{d, b, c\}$
300	$\{d, e\}$	$\{d, e\}$
400	$\{a, c, d, e\}$	$\{d, e, c\}$
500	$\{a, b, e\}$	$\{e, b\}$

Table 5.3: Example transaction set $\mathcal{T}$

The term *frequent pattern* is synonymous with *frequent itemset* in the definition we gave above. An example FP-tree is shown in figure 5.10. The frequent single item sets (given a minimum support count of 3) and their support count in table 5.3 are, in descending frequency order:

$$(d : 4), (e : 4), (b : 3), (c : 3)$$

Generating the frequent one-item sets and ordering them is the first step in growing an FP-tree. The next step is making another scan of the transaction set, to find the ordered frequent items for every transaction. From this ordering, a tree can be generated. The tree combined with a header table forms the FP-tree.

From a FP-tree, the frequent patterns or itemsets can be extracted. For every node in the tree, following its linked nodes leads to the set of frequent patterns that contain that node. The subset $x_1, \ldots, x_n - 1$ for the nth item in a path downwards the FP-tree is called the *prefix path* of node n. For example, c appears in three paths:

$$\langle (d : 4), (e : 3), (c : 1) \rangle$$

$$\langle (d : 4), (e : 3), (b : 1), (c : 1) \rangle$$

$$\langle (d:4), (b:1), (c:1) \rangle$$

In the first path, the conditional pattern base for c is $\langle (d:4), (e:3) \rangle$. That means that the frequency of the pattern $d-e-c$ is at most 1, the frequency of node c, or the last node in c's prefix path. The set of all prefix paths for a node n is called the *conditional pattern base* for that node. For node c, the conditional pattern base is $\{\langle (d:1), (e:1) \rangle, \langle (d:1), (e:1), (b:1) \rangle, \langle (d:1), (b:1) \rangle\}$ Notice that the count of the items in the paths of the pattern base is at most that of node c.

A new FP-tree is grown from this conditional pattern base, where only the items are included that are in every path of the pattern base. This leads to the construction of a tree with $(d:3)$ as it's only branch, which means that dc is a frequent pattern with support count 3. This is repeated for the other items in the header table, in sequence from least support to most (so $c-b-e-d$) to produce all the frequent itemsets in the FP-tree.

Performance FP-growth performs especially well compared to Apriori for low *minsup*, since Apriori needs to create very large one- and two-sized frequent itemsets. Even for higher support thresholds, FP-growth consistently outperforms Apriori [24].

5.1.3.7 Summary of association mining techniques

The three techniques *Apriori*,*partitioning*,*sampling*, *hashing* and *FP-growth* that are based on the principles described in Apriori [2] all have better performance in a subset of cases. Partitioning and FP-growth are faster in the cases presented by [24][35]. Sampling is also a potentially fast technique, as it reduces the set of candidates considerably, at the cost of completeness [3]. As this is an undesirable feature, the sampling technique will not be considered any further. Hashing is faster than Apriori, but only when the support threshold is set high enough [31].

From this we conclude that the algorithms of choice for assocation rule mining are Apriori, partitioning and FP-growth. The choice of algorithm depends on how fast it needs to be for the specific case, but all three will lead to a complete set of association rules.

5.2 Process elicitation

Figure 5.11 shows the procedure behind process elicitation. It starts with a set of events as they occurred in the past, including their starting and ending time. This is called an event log. These logs are processed by some method and result in an explicit *actual* process, as it can be perceived from the data.

5.2.1 Process mining

Process mining is an automated way of extracting workflow patterns from data. This data–the event log– is a set of activities with the timestamp on which they occurred and some label for each activity instance. An example event log is shown in table 5.4. A possible process that these activities compose is shown in figure 5.12. Process mining is commonly used to control business processes by making sure the actual process model – induced from data – matches the

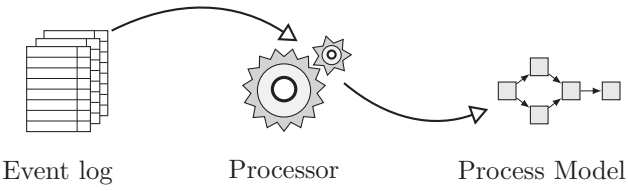


Figure 5.11: Process mining: Basic procedure

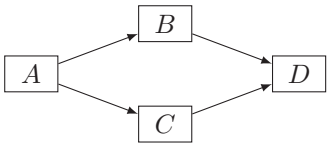


Figure 5.12: A possible process from the event log in table 5.4

designed process model. Another application is discovering the process if no process model exists, or if it is outdated. Van der Aalst *et al.* refer to the first as *delta analysis* and to the latter as *process discovery* [41] [38].

This section starts with some more formal definitions for the terminology used, before algorithms for process mining are discussed.

Definitions The central terms in process mining are *process*, *activity*, *event log* and *work flow*. Some of the terminology used in section 5.1.3 also applies to process mining.

A process P is a directed graph where each node represents an activity and each activity $A \in P$ has at least 1 incoming or outgoing edge. A process has one *start* and *end* activity, that respectively only have outgoing and incoming edges. Formally [1]: a process consists of a set of activities

$$A_p = \{A_1, \ldots, A_n | 1 \leq i \leq n \wedge A_i \in \mathcal{A}\}$$

a directed graph

$$G_p = (A_p, E_p)$$

an output function

$$0_p : A_p \rightarrow \mathcal{N}^k$$

and a Boolean function for all edges

$$\forall (u, v) \in E_p : f_{(u,v)} : \mathcal{N}^k \rightarrow \{0, 1\}$$

An activity is an atomic item of work, which has a label L. An event log is a set of tuples that consist of minimally a label and a timestamp. Some authors include a case identifier as well, referring to the process instance to which a tuple belongs. Formally, an event log is a set of traces $\{\sigma_1, \ldots, \sigma_n\} \subseteq \mathcal{A}^*$. We use the term work flow synonymous with process.

Algorithms One group of algorithms for process mining shares some of the characteristics of association rule mining techniques discussed in section 5.12. The algorithms aim to find rules such as $A > B$, with the difference that rules are not (merely) correlation, but are dependencies in process mining. Generally, we can say that process mining is a sub field of association rule mining that incorporates knowledge about the domain (a process has a start and an end, every node is an activity, etc.) into its algorithms.

The example rule means B follows A, B happens parallel to A or any other process relation. Most commonly, the *follows*, *preceeds*, *parallel* relations are described in process mining algorithms. What sets process mining algorithms apart is what assumptions they need about the process (mostly: if the algorithm can find cycles) and the quality of data (some algorithms cannot cope with data noise) [1] [12] [20] [22] [37] [38] [39] [42] [41].

Desired features of process mining algorithms Agrawal *et al.* give three conditions that the process model obtained by using some process mining algorithm ideally satisfies: *completeness*, *irredundancy* and *minimality* [1].

- *Completeness* means that the process model contains all the dependencies that appear in the log and that the executions in the log are possible in the model. The latter is a relatively easy check of the found process model against the event log, but the former part is more difficult to satisfy. The problem is that dependencies are implicit in the log. One way to be sure that the process model is complete is assuming that every node is dependent on every other node – include all (possible permutations of) $\{A \Rightarrow A' | A, A' \in \mathcal{A} \wedge A \neq A'\}$, where $\mathcal{A}$ is the set of activities in the log. This approach will likely violate the other desirable conditions.

- *Irredundancy* means that the process model should not add dependencies that "aren't there". To check if a dependency is in fact not present in the data, we can see if there is some combination of activities from the log that satisfy the dependency. For example, the path $A \overset{>}{\rightarrow} B \overset{>}{\rightarrow} D$ is present in the process model in figure 5.12, where $A \overset{>}{\rightarrow} A'$ means that the execution of activity A preceeds activity A'. This dependency is not redundant, as the event log shows that A sometimes occurs before B and B sometimes occurs before C.

- *Minimality* means a restriction on the visual representation of the process model: the graph representing the process should have an minimal number of edges.

We will use these three algorithm features as a means to compare process mining procedures, together with the limitations of those algorithms.

Process mining limitations The definition of a work flow (which we use synonymously with process) is interpreted differently by different manufacturers of workflow management systems and in different workflow languages [28]. As a consequence, algorithms that aim to find workflows matching some workflow language will have different interpretations of constructs such as splits and joins in the flow. Also, almost half of the workflow languages discussed in [28] do not allow for cycles. For an algorithm based on such an "acyclic" language

this means that it will not include cycles in the model, even if they do exist as traces in the log. This is especially an issue of formalism, as it is sometimes considered undesirable for a process to deadlock. If the workflow language used does not have loops, it is safe in the formal sense of the word. For example the procedure described in [1] – based on IBM's MQSeries/Workflow (then known as Flowmaker) – will only find process models without cycles/loops.

A different limitation altogether is dealing with noise. For example, events may not have been logged or have been logged incorrectly – as the wrong activity, or with the wrong timestamp. The algorithm will have to have some way of detecting and subsequently removing noisy traces in the event log. If an algorithm cannot deal with noise, this is a limitation. Related to noise is overfitting, or the condition where exceptional traces unwantedly make their way into the model, due to an event log that is too large.

Metrics As mentioned briefly before, the process mining paradigm is an specialization of the association rule mining paradigm that adds domain-specific metrics to the relation of two items(activities). Some of the most common metrics [40] are discussed below.

- *Following* An activity B is said to follow A – denoted $A \overset{>}{\rightarrow} B$ – iff there exists some trace $\sigma_1, \ldots, \sigma_n \in \mathcal{A}^*$ where $\sigma_i = A$ and $\sigma_{i+1} = B$.

- *Causality* There is a causality relation $A \rightarrow B$ iff $A \overset{>}{\rightarrow} B$ and $B \overset{\not>}{\rightarrow} A$. This means that A always comes before B and we know that in the process model an outgoing edge of the node labeled A leads to the node labeled B.

- *Parallel* A and B are said to be parallel ($A \parallel B$) if $A \overset{>}{\rightarrow} B$ and $B \overset{>}{\rightarrow} A$. The rationale is that if for two activities the one sometimes occurs before, sometimes after the other, they are in fact likely to be executed parallelly.

- *Choice* $A \vee_{\#} B$ means that either A or B occurs: $A \overset{\not>}{\rightarrow} B$ and $B \overset{\not>}{\rightarrow} A$. Therefore, a relation is only classified as choice if A and B always occur at (exactly) the same time. This is of course natural for choice: having completed step A' at time t a worker starts either A or B at time $t + 1$.

It's these metrics that are used to specify the relations between the activities in the event log. For example, looking at table 5.4 and figure 5.12, we see that the trace for case 1 is A, B, C, D and for case 2 its A, C, B, D. Therefore, we conclude that $A \overset{>}{\rightarrow} B$ in both case 1 and 2. Also, $B \overset{\not>}{\rightarrow} A$, therefore there is a causal relation: $A \rightarrow B$. $B \overset{>}{\rightarrow} C$ as well as $C \overset{>}{\rightarrow} B$, for different cases. Therefore, $A \parallel B$. Other relations that can be induced are $A \rightarrow C$, $B \rightarrow D$ and $C \rightarrow D$. Together, these relations lead to the process model in figure 5.12.

Finding the relations between activities is done through algorithms. Not all of the algorithms include all of the metrics, as mentioned in the paragraph "Limitations", some workflow languages (and related process mining tools) do not support certain relations (specifically relations such as $A \rightarrow A$).

The next sections discuss algorithms to produce process models from event logs.

Case	*Activity*	*Timestamp*
1	A	00 : 00
2	A	01 : 00
1	B	01 : 30
2	B	01 : 35
1	C	01 : 30
2	C	01 : 40
1	D	02 : 15
2	D	03 : 15

Table 5.4: Example event log

α-algorithm Van der Aalst *et al.* have proposed the α-algorithm for process mining. This algorithm can be used to model any type of workflow, as long as it is a sound workflow. The assumption for α is, in other words, that the process/workflow represented in the event log:

- *has no deadlocks*, or that $\forall\{\sigma_1, \ldots, \sigma_n\} \subseteq \mathcal{A}^* : \sigma_n = A_{end}$. This means nothing "bad" happens.
- *has no dead transitions*. This means that something "good" eventually happens.
- *there is an "option to complete"*. This means that for every node, the end state can be reached.

The algorithm consists of eight steps, takes an event log $W \subseteq \mathcal{A}^*$ as input and produces a Petri net P_W, T_W, F_W as output [2]. Figure 5.13 shows the Petri net resulting from a run of the α-algorithm on the event log in table 5.4.

Given the three desirable features of a process mining algorithm presented above, we can now see how (this instance of) the α-algorithm compares. First, completeness: the traces A, B, D and A, C, D can be derived from the produced Petri net. Therefore, the algorithm's result was complete for this instance. Second, irredundancy: only the traces A, B, D and A, C, D are possible in the resulting Petri net, therefore there was no redundancy. Finally, minimality: There are no edges in the Petri net that are redundant, so it is minimal.

The α algorithm has been shown to have all three desirable characteristics for this case. The question is, if it has those characteristics for any event log.

Limitations of α The α algorithm does not deal well with short loops or noise [41]. For example, if activity A is repeated, represented by a trace A, A, B, C, this will not lead to the correct representation shown in figure 5.14b. Instead, the algorithm will discard the extra A, as it will not identify the relation $A - A$, as can be seen in step 4: there no "choice" between A and A. There is a choice between A and B, so the relation $A - B$ will remain. The incorrect interpretation is shown in figure 5.14b. Where the looped trace longer, such as a trace $A - B - C - A - D$, the algorithm would give the right interpretation.

[2] For an explanation of Petri-nets, see Desel *et al.* [16]

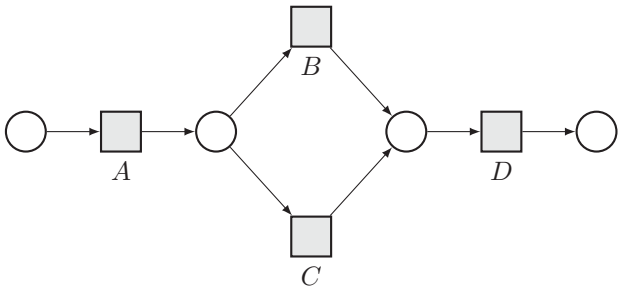
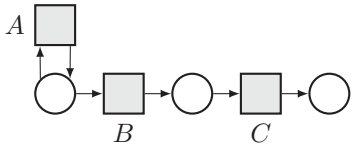
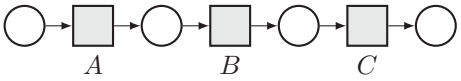


Figure 5.13: A Petri-net generated from table 5.4 by the α-algorithm



(a) Right result



(b) Wrong result

Figure 5.14: Comparison of wrong and right result for the α-algorithm

In the words of Van der Aalst *et al.* the algorithm cannot deal with "arbitrarily short loops" [41].

A different problem altogether is how α deals with noise. Consider a process where activity B follows A, leading to the trace $A - B$ appearing in the log. At some point, due to a registration error, the trace $B - A$ was entered in the log. If this trace occurs 1 times in the log and trace $A - B$ occurs one thousand times, the algorithm will still conclude that there is a relation $B \overset{>}{\rightarrow} A$. More precisely, as $A \overset{>}{\rightarrow} B$, the relation will actually be $A \parallel B$.

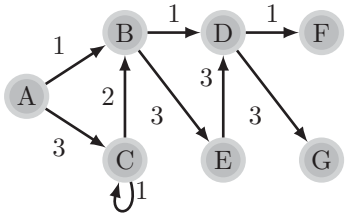
In conclusion, we can say that α is usable only if we know the data to be noiseless and that either there are no "arbitrarily small" loops or that –for whatever reason– it does not matter that these are ignored.

Other algorithms To address the problem of noise, mentioned in the previous paragraph, several modifications of α have appeared. One of them, "Little Thumb" uses heuristics to deal with noise: it considers the ratio of $A \overset{>}{\rightarrow} B$ to $B \overset{>}{\rightarrow} A$ to find the actual relation between the two activities. This augmentation of the α-algorithm is however still limited, as its heuristic approach only works for the causality relation, not for parallelism or choice [44]. Also, Little Thumb cannot find short loops, just as α.

A (somewhat) different approach to α is *InWoLve*, which uses a stochastic approach to extract the process from an event log. What this means is that it

#	*Trace*
1.	$\{A, B, E, D, G\}$
2.	$\{A, C, C\}$
3.	$\{B, E, D, F\}$
4.	$\{A, C, B, D, G\}$
5.	$\{B, D, G\}$
6.	$\{A, C, B, E, D, G\}$

(a) Example set of traces



(b) Elicitated graph

Figure 5.15: Converting traces to a graph

takes the probability of one node preceeding another into account to determine the process model. This is clearly closely similar to the heuristic approach of Little Thumb, to the extent that InWoLve also has difficulty identifying concurrency [27]. InWoLve performs even worse than the α-variants when it comes to loops, as the algorithm removes any duplicate activities from traces.

From a comparison in [40] we can conclude that dealing with loops in processes is a problem for existing algorithms. Furthermore, the majority of algorithms takes the assumption that there is no noise in the event log. This leads us to conclude that Little Thumb and InWoLve are the most likely choice as a process mining algorithm, as they at least deal with noise – something to expect when using the algorithm in "real" situations.

The discussed algorithms have been combined into a framework of their own, called ProM [43]. Besides implementations of α and the other algorithms, it includes tools to data import and visualization. We will refer to this toolset from now on when talking about process mining to indicate that one of the algorithms discussed above is to be used.

5.2.2 Shortest-path approaches

As noted in the previous section, none of the existing process mining algorithms is able to deal well with loops or noise. Also, considering the goal of the literature review is to find methods for finding a process graph and predicting a path through that graph, it is important to consider the graph elicitation as an intermediate step, not an end product.

Looking only at the need for a complete and accurate graph, a possible approach is to take the event log and create a graph that contains every edge in every trace of the log. Consider the set of traces in table 5.15a, converted to the graph in figure 5.15b. Every combination of two subsequent activities is an edge in the graph, every activity is a node. There are no duplicate edges or nodes, but the number of occurrences of two nodes subsequently is noted as the weight of the edge.

Since every node and edge from the traces (under the above definition of a node and edge) is represented by exactly one node/edge in the graph, this naive graph elicitation algorithm is complete by the definitions of chapter 3. The algorithm is naive, as it also includes possibly noisy edges, that only occur

a few times in the data set. In figure 5.15b, the $C - C$ edge could be noise, it occurs only once in the data. This makes the naive graph elicitation algorithm unaccurate. To remedy this flaw inherent to the naive approach, the weights of the nodes can be considered as the likelihood of an edge *not* being noise. The more times an edge occurs, the less likely it is to be noise.

Not only individual edges are noisy, also larger sets of edges might be noise. This would result in several nodes connected by low-weight edges. This points to a possible solution: use a path algorithm to determine paths that have the heighest weight, or to exclude the paths with the lowest weight. This type of algorithm is known as a shortest-path-algorithm: they find a path through the graph that has the lowest possible weight [8].

The most well-known shortest-path-algorithm is Dijkstra's algorithm, which calculates the single shortest path through a series of nodes. This algorithm can be used to determine the distances from a start node to every other node. A distance is the compound weight of the edges towards a node. For example, a distance between A and G is 5. This is also the path that has the lowest compound weight for its edges. Conversely, Dijkstra's algorithm could also be modified to determine the path that has the highest weight for all its edges combined. This would represent the most likely path. A problem that would occur with this approach is that loops always lead to a higher weight when iterated once more. This can be remedied by relabeling duplicate activities within a trace so that they are unique. In figure 5.15b, this means that C is replaced by C and C', $C - C$ is replaced by $C - C'$.

There are some problems when using Dijkstra: the algorithm has to be run for every node in the graph to give the shortest or longest path from the start node to some other node. Second: Dijkstra cannot give the shortest path between two specific nodes, it only gives the node that has the shortest/heighest distance from "start". A different shortest-path algorithm, known as Floyd-Warshall does have this possibility: it gives every shortest path between two nodes. This makes it possible to find the most likely path or most unlikely path from one node to another.

5.2.3 Summary of process elicitation techniques

The α-algoritm, InWolVe, Little Thumb and the other algorithms all have difficulty with dealing with noise and loops in the data. As both these elements will likely occur in data, we have proposed a more naive graph elicitation algorithm, that includes all nodes and edges present in the event log traces. As this combination of a naive graph elicitation algorithm and a shortest path algorithm combines an ability to deal with noise with an ability to deal with loops, we consider it more suitable for our purposes than the process mining algorithms mentioned above. This especially applies to the fact that the process graph is an intermediate step: the proposed combination delivers a complete process graph and provides a way to determine the most likely path between nodes. If we can predict the node the path will end at (using the correlation induction), this provides a complete solution to the first three research questions.

5.3 Conclusions

This chapter has set out to find answers to three questions:

- What methods exist for finding a process graph?
- What methods exist for finding a subsequent path?
- What methods exist for finding path-variable correlations?

As mentioned in the last section, there is a group of methods known as process mining techniques that make it possible to induce a process graph from data. From these techniques, a naive graph elication technique combined with the Floyd-Warshall algorithm is the most suitable for our purposes, especially so since it not only answers the first research question, but is a first move to answering the second question.

For finding the relation between variables, or any other element, the correlation induction section concludes that random forests and boosting are the most accurate techniques. This makes them the methods of choice for finding path-variable correlations, as well as relations between elements or subsets of the path. Also, association rule mining might be used to find sets of nodes that occur together in the data.

Combining these findings, we conclude that a method for finding a subsequent path from some "node" onwards could be to use a classifier to predict the end node of the path and consequently using Floyd-Warshall to find the most likely path between node and end node. Alternatively, association rule mining might be used to predict the subsequent path from the set of traversed nodes, if there is a rule that matches the traversed nodes.

Chapter 6

Hospital Case

The literature research has lead to several techniques that can be used to answer a business question that is related to a process graph. This chapter describes how those techniques are used to solve the hospital case, that has been the driver for this research. The aim of the chapter is to show how the methods perform in practice and how they should be combined.

As the research aim is to use data to answer business questions, the chapter starts with a description of the data relevant to the hospital case. Next the tools and experiment setup are discussed. The experiment is aimed at testing the methods for the hospital case, to see if they can answer the hospital's business question. We conclude the chapter with a discussion of the results of the experiment

6.1 Data

The business question of the hospital is "When will there be a cash flow for a care path and how large will that cash flow be?". From the introduction, we know that the care product decides the declarable amount for a sub path and that the sub path will become declarable some time after the final activity. This means that the data we want to use is a set of finished sub paths, including the care product and duration of the path.

The available data consists of a 70GB database of patient care data. There is a table for performed activities and sub paths, the two elements that are most relevant to the case. The structure of these two tables in terms of the relevant columns is shown in tables 6.1 and 6.2.

As stated in the introduction of this research, it is the activities performed that decide the care product. This does not mean that every activity influences the care product, in fact, laboratory work and medicines have no influence on the grouper result . Therefore, the test data will omit those activities which activity codes represent lab work or medicine. Looking at the two tables, the columns of most interest are the "Care activity code" and "Date" in the activity table and the "Care product", "Diagnosis code", "Care product", "Start date" and "End date" in the sub path table.

To produce a data set usable for an experiment, the activity and sub path tables where joined to get a new table with the sub path id, care product, activity

code and registration date. A separate table was made for every diagnosis, as the diagnosis is not part of prediction, but is known at the start of a sub path. Also avoiding overly large tables helps the speed of testing the data and process mining algorithms. The produced tables where converted to a *Sub path id* − *Care product* − *Activity 1* − ... − *Activity n* format, as classifiers require all relevant fields to occur in a single tuple.

The data sets chosen –that is, the diagnoses chosen– are two cardiology related and one lung medicine related: Angina pectoris (chest pain), pericarditis (an inflammation of the heart) and malignant throat cancer. These sets where chosen for three reasons: they are (relatively) large sets, ranging from 300 to 9000 tuples, they have distinctive care activities and care products, and they belong to the medical department Topicus is currently working for. Since the DOT methodology will not be in effect until January 2012, all data in the database is converted data from the current DBC methodology. Although this does lead to more noise in the sense of missing data –DOT uses more fields than DBC in some cases –, the data is from actual performed care and has been run through the grouper.

Description of the data The three data sets are named according to the specialty and diagnosis code they belong to: 320.202 for angina pectoris, 320.701 for pericarditis and 302.66 for malignant throat cancer. 320 stands for the cardiology specialty and 302 for the lung specialty.

The 320.202 data set consists of 9950 rows in *Sub path id* − *Care product* − *Activity 1* − ... − *Activity n* format, 320.701 consists of 274 rows and 302.66 of 1985. There are 150 unique activity codes in 320.202, 320.701 has –even though it has less rows– 172 unique activity codes. 302.66 has 102 unique activity codes. There are 22, 6 and 11 care products associated with the sub paths in 320.202,320.701 and 302.66, respectively. The longest sub paths (highest number of activities) are length 152, 174 and 104 for 320.202,320.701 and 302.66. The average path length for the data sets is 6, 11 and 11.

To summarize, the 320.202 data set is large in terms of rows, but has the relative lowest number of unique activities and length of sub path. The 320.701 is the smallest set, but has long sub paths and a high number of unique activities. On the other hand, it has few care products. The 302.66 data set is somewhere in between.

Data noise As mentioned in chapter 5, noise in the data will affect some classifiers negatively. The difference between classifiers is the ratio of noise to actual data they can deal with. The amount of noise in the three data sets is high, due to how the data was recorded. The data sets have activity codes labeled activity 1 through n, based on the date on which they were registered. The trouble is that activities are not registered on a daily basis, resulting in multiple activities per day per sub path. Sequencing the activities that occur on one day results in noise, as there is no way of knowing the order in which the activities were actually performed. This noise will likely hinder the accuracy of the classifiers, as the occurence of multiple activities on a single day is very common.

Activity id	*Care path id*	*Sub path id*	*Care activity code*
The unique id of the performed activity	Unique identifier of the care path containing the sub path the activity is part of	The identifier of the sub path the activity is part of	A (non-unique) code describing the activity
(cont.d)	*Date*	*Performing specialty*	*Amount*
	The date on which the activity was registered	The specialty performing the activity	The quantity of the activity, for example used for medication

Table 6.1: The performed activity table

Sub path id	*Care path id*	*Start date*	*End date*	*Medical specialty*
The unique identifier of the sub path	The care path the sub path is part of	The date on which the sub path was opened	The date on which the sub path was closed	The specialty responsible for this sub path
(cont.d)	*Care type*	*Diagnosis code*	*Care product code*	*Closure reason*
	The care type of the sub path. See appendix A.	A code for the diagnosis that led to this sub path	The care product for the closed sub path	The reason the care path was closed, for example, enough time had expired.

Table 6.2: The sub path table

6.2 Experiment

The data sets available determine the type of experiment that can be done on the data. The goal of the experiment is to find the best performing technique for predicting the care product –and therefore the declarable amount for care–, and to find exactly how well this technique performs. Also, the experiment is designed to find the most suitable method for predicting the duration of the sub path.

The main techniques tested are random forests and boosting, as these were found to be the theoretically most accurate techniques. As a comparison, decision trees, kNN clustering and naive bayesian classification are shown as well. These techniques were found to be less accurate in theory, therefore they give some perspective to compare the best classifiers to.

Design of the experiment The research questions mention finding a subsequent path, finding path-variable relations and finding a process graph. In the context of the hospital data, a subsequent path is the set of activities that are performed after some activity n. The path-variable relations of interest are the relation *set of activities* $\rightarrow$ *careproduct* and *set of activities* $\rightarrow$ *total duration of sub path*. Finding the process graph means establishing which activities follow eachother.

As the research aim is to predict the activities still to come, the experiment tests how well we can product the care product from the first activity, the first two activities, the first three activities and so on. Also, there is a test to predict the activity $n+1$ from the first n activities, to see if this is an accurate way of finding the subsequent path. Furthermore, given a (predicted) care product, we test the Floyd-Warshall algorithm to find the most likely path between the two, if we consider the care products to be the final nodes on the graph. The list below shows all performed tests:

Test 1.) Predict the *care product* from *activity* $1 \ldots n$, for $n \in \{1 \ldots 30\}$.

- a.) Predict *declarable amount* from *activity* $1 \ldots n$, for $n \in \{1 \ldots 30\}$.
- b.) Predict *duration* from *activity* $1 \ldots n$, for $n \in \{1 \ldots 30\}$.

Test 2.) Predict *activity* $n+1$ from *activity* $1 \ldots n$, for $n \in \{1 \ldots 30\}$.

Test 3.) Predict *activity* $n+1 \ldots$ *care product* from *activity* n.

- a.) Predict *duration* from *activity 1 ... care product*.

Test 4.) Predict *activity* $n+1 \ldots <<$ *final activity* $>>$ from *activity* n.

The fourth test in this list is a test to predict the path using only the current activity, and seeing what the final node is that is the most likely. Again, the final nodes in the graph are care products, so this is a test for both finding the subsequent path and careproduct. This test uses the Dijkstra algorithm, as this is more suitable for finding the single most likely path than the Floyd-Warshall algorithm (see chapter 5). Tests one (and its sub tests) as well as test two will survey the classifiers mentioned at the start of this section– random forests, boosting, kNN and naive bayesian classification– and will rank them for accuracy.

Hypotheses Parallel to the tests described above, there are hypotheses we want to test with the experiment. These hypotheses describe the feasability of prediction with the methods described in this research. If they are rejected, that part of the approach should be considered infeasable for the hospital case. The hypotheses are:

Hypothesis 1.) It is possible to predict the care product with greater accuracy than $\frac{1}{|\{care\ product\}|}$

a.) It is possible to predict the care product with greater accuracy than $\frac{1}{|most\ frequent\ care\ product|}$

Hypothesis 2.) Prediction gets better with more activities performed.

a.) Prediction of the care product gets better with more activities performed.

b.) Prediction of the declarable amount gets better with more activities performed.

c.) Prediction of the duration gets better with more activities performed.

Hypothesis 3.) It is possible to predict the duration with a smaller standard deviation than the standard deviation of all durations in the data set.

These hypotheses are all closely related to the tests described earlier, but allow for drawing more clear conclusions as to how usable the methods and approach discussed in this research are to the hospital case.

6.3 Tools

In order to perform an experiment, we need tools that work on the data. There is a large variety of tools on the market for data analysis and data mining, often integrated in business intelligence packages. The main requirements for a data mining tool are that it has implementations for the selected algorithms and that it provides functionality to analyze the results of data mining. Also, a data mining tool preferably manages data import and conversion.

The tools we investigated for use are the Java-based Weka and *R*, a combination of a statistical analysis tool and programming language [34]. Initial experiments have shown that Weka runs into memory issues when handling large data sets and complex algorithms, such as random forests. *R* does not have these problems, and provides an implementation for all the algorithms used in the experiment. Furthermore, *R* has the statistical tools to analyze the results of data mining. This is especially relevant for things such as calculating the standard deviation of the time-experiment results. This makes *R* the tool of choice for the data mining part of the experiment.

Besides the software tool, the choice of algorithm is of course important, as there may be multiple implementations of an algorithm, especially for decision trees. The decision tree algorithm we use is CART (classification and regression

trees), as implemented through the *rpart* (recursive partitioning)-package of *R*. This package can also be used to do C4.5 like tree growing by selection of the split measure (Gini versus information-based). The random forests algorithm is implemented in the *randomForest* package, and can be used both for classification and regression analysis – used for predicting duration. Boosting is tested with the AdaBoost.M1 algorithm, as this is the only AdaBoost-implementation available in *R* through the *adabag* package. *k* Nearest Neighbors is implemented in the *e1071* package. Finally, naive bayesian classification uses the implementation provided by the *R* core.

For predicting a path through a graph, we have implemented a simple graph in Java. Also, an implementation of the Floyd-Warshall algorithm was adapted from an example algorithm sourced from [6]. An implementation of the Dijkstra shortest path algorithm was derived from [8].

6.4 Results

This section describes the results of the experiment presented above. As mentioned before, the experiment is in essence a statistical experiment, which means that its results are also statistical. For continuous variable prediction, such as the duration of care, this means that a result is the average error. For classification, the result is the number of correctly classified instances as a fraction of the test set.

Below, the results for the tests from the "Experiment" section are presented. Where applicable, we will also discuss the impact the result has on the hypotheses.

6.4.1 Predict the care product from activity 1...n

Table 6.3 shows the results for this first test: classifying a section of the sub path as the care product. The results shown use a random forest of 2500 trees and a training set of 50%. This has been shown to be the most accurate configuration for the classifiers: a test with 10% training set and 1000 trees yielded $0.3971970, 0.4231762, 0.3361323$ for 320.202,320.701 and 302.66, respectively.

What the table shows is that the random forest method is *on average* the most accurate classifier of the five tested. Note that accuracy is the fraction of correctly classified instances from the total number of classified instances. Just as was found in literature, boosting is the next best thing. *k*NN and Naive Bayes are the worst two classifiers in terms of accuracy, but their place differs for the three data sets. Notably, the maximum accuracy of the naive bayesian approach for the second data set is the highest of all five methods. This is a result of the set up of the experiment: the test is performed for sub sets of sub paths of length 1 through 30.

For the larger sizes of sub sets, the number of sub paths of that length decreases rapidly: there are 9889 rows for sub paths of length 1 (training set plus test set), there are only 80 rows for sub paths of length 30 in 320.202. The total size of the data set is 9950 rows, but some have zero activities. This is the result of the removal of non-relevant activities, such as laboratory work and medication. Figure 6.1 shows the relation of the number of activities considered and the available number of sub paths of (at least) that length.

The influence of the number of activities considered on the accuracy of the classifiers is shown in figure 6.2. To avoid confusion, only the accuracy of the random forest classifier is shown. What the plot shows is that the performance of the classifier increases until around 6 activities included, then decreases as the number of test cases decreases, and becomes somewhat erratic once the number of activities is around 16. This is likely caused by the low number of cases for that amount of activities (around 400 in total). The accuracy is lowest for the 302.66 data set and highest (but also most erratic) for 320.701. This is explained by the low number of rows for 302.701. There is however a peak for accuracy for each data set around 6 activities. This could be explained by it having the optimal ratio of number of activities included to the number of cases (size of test set). The plot also shows the threshold values for the three datasets, that is: the accuracy of predicting that the careproduct will be the careproduct that is most frequent in the dataset. For dataset 320.202 its 0.21 (not visible in the plot's range), for 320.701 its 0.41 (red dashed line) and for 302.66 its 0.35 (blue dashed line). Interesting to see is that only the accuracy for the 302.66 dataset falls below the threshold at some point.

To see the influence of the number of rows for a certain set of activities, table 6.4 shows a test for a subset of the data. This subset contains only the rows that have at least ten activities. For this subset, we test to predict the care product for number of activities $1 \ldots 10$. This way, there is an equal number of rows for every number of activities considered, so that the test shows only the influence of the number of activities on the accuracy of the prediction. More interesting than the table, which shows similar results to table 6.3, is the plot in figure 6.3. This shows that there is only a slight increase in accuracy with more activities included for two out of three datasets. Only for the 320.202 dataset is there a clear increase in performance and a steady rise with the increase in number of activities. The other two show a drop in accuracy between two and eight activities, but do have a slightly higher accuracy for activity ten than for activity one. Perhaps the datasets' size (2297 rows for 320.202, 99 for 320.701 and 831 for 302.66) again causes the irregular nature of the accuracy, but the size for 66 and 202 is large enough to suggest otherwise. As there are on average 11 activities for 302.66 and 320.701, we would expect the accuracy to be at its best for around 10 activities.

Predict declarable amount from activity 1...n When predicting the declarable amount, there is an intermediate step: predicting the care product. Second, the declarable amount for that care product is looked up in a table. Table 6.5 shows the results for test $1a$. The test results represent the difference between actual total declarable amount and predicted total declarable amount, as a percentage of the total declarable amount. The lower the percentage, the better the classification.

Surprisingly, the results are quite different from the care product prediction test. CART, Random Forests and kNN are the best classifier, each for a different data set. Furthermore, Naive Bayes is consequently significantly worse than any other classifier. This suggests that a wrong prediction of the classifier actually comes in gradations: some wrong predictions are further off than others. This is illustrated by the 30% accuracy displayed earlier by Naive Bayes, versus 15-40% error for this test. The fact that there is a "wrongness" to a wrong prediction

Training set: 50%. Number of trees in random forest: 2500.

320.202				
	Min	Max	Mean	Median
Random Forest	0.30	0.69	0.47	0.44
CART Tree	0.24	0.49	0.37	0.35
Adaboost.M1	0.24	0.59	0.41	0.40
Naive Bayes	0.21	0.45	0.30	0.28
k Nearest Neighbour	0.20	0.53	0.31	0.28
320.701				
	Min	Max	Mean	Median
Random Forest	0.44	0.71	0.55	0.53
CART Tree	0.00	0.75	0.47	0.50
Adaboost.M1	0.11	0.75	0.52	0.56
Naive Bayes	0.00	0.80	0.45	0.45
k Nearest Neighbour	0.11	0.63	0.42	0.43
302.66				
	Min	Max	Mean	Median
Random Forest	0.28	0.55	0.40	0.38
CART Tree	0.16	0.55	0.35	0.33
Adaboost.M1	0.24	0.56	0.38	0.35
Naive Bayes	0.19	0.46	0.31	0.29
k Nearest Neighbour	0.20	0.45	0.28	0.25

Table 6.3: Prediction of care product from activity $1 \ldots n$

	Min	Max	Mean	Median
320.202	0.43	0.58	0.49	0.50
320.701	0.36	0.54	0.43	0.40
302.66	0.43	0.48	0.45	0.45

Table 6.4: Prediction of care product from activitiy $1 \ldots 10$

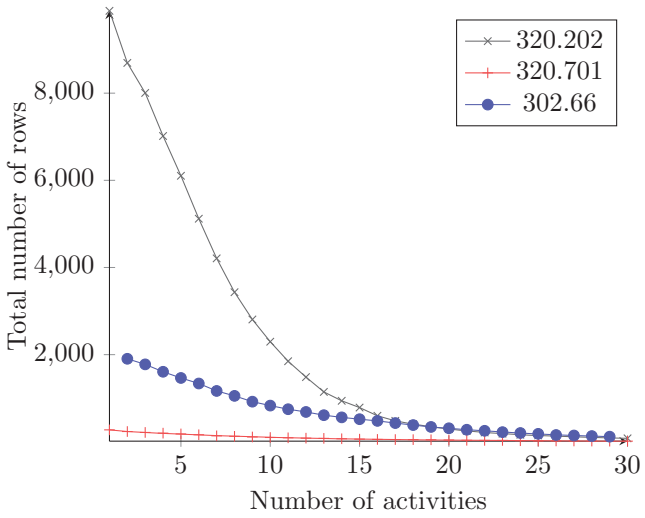


Figure 6.1: Number of rows for number of activities $1 \dots n$

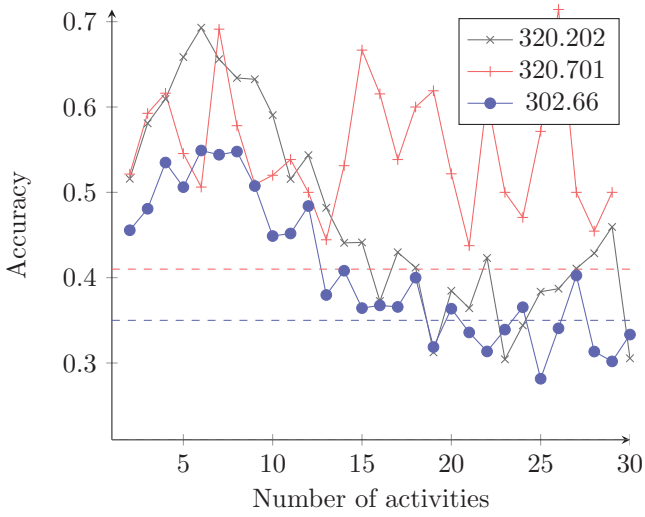


Figure 6.2: Random Forest accuracy for number of activities $1 \dots n$

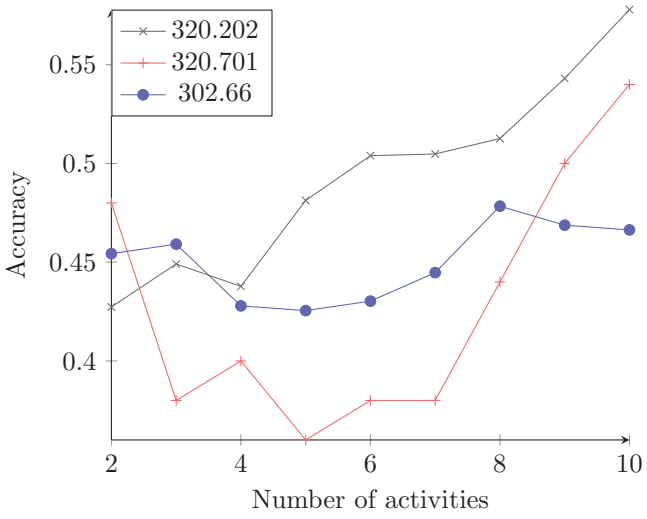


Figure 6.3: Random Forest accuracy for number of activities 1 . . . 10

can be traced to the grouper (a binary tree) underlying the actual classification. It is possible to predict another care product that is in the same grouper sub tree or branch, that would be fewer off –in terms of declarable amount– than a care product further removed in the tree. The independence assumption of Naive Bayes somehow results in higher errors.

More importantly, what the results point out is that the error is lower than the inaccuracy of the care product classification. Whereas 40 to 60% is misclassified, the error is between 6 and 17%. This is the result of law of large numbers: given a sufficiently large amount of cases, the positive and negative errors will even out. That it is not purely the size of the data set that determines the error: the largest (320.20) and smallest (320.701) data set have the lowest error.

The error rate for different number of activities included is shown in the plot in figure 6.4. Again, only the Random Forest error rates are shown, as this is the most consistent performer of the five techniques. Remarkably, the average absolute error per case (in euros) only marginally changes with the number of activities for data sets 320.202 and 320.701, but increases exponentially for 302.66. A possible explanation for this fact is that the standard deviation of the declarable amounts in 302.66 is greater than for the other two data sets.

Predict duration from activity 1. . . n As predicting the duration of a care path appears to be the most difficult to predict (there is no fixed relation care product-duration, as opposed to declarable amount, which follows from care product), we have first researched a best case scenario for feasibility. This means using the best classifier from the first tests –Random Forests– and making the test set as large as possible in terms of activities included, as this theoretically increases performance.

The test set used consists of all activities in the data set. To avoid classifier errors occurring on missing data (not every sub path has the same amount of activities), the data sets are converted so that they have the counts of an activity in every column. If an activity does not occur in that sub path, the

320.202				
	Best	Worst	Mean	Median
Random Forest	0.00	0.15	0.07	0.06
CART Tree	0.00	0.21	0.07	0.07
Adaboost.M1	0.00	0.20	0.08	0.07
Naive Bayes	0.16	0.62	0.33	0.25
k Nearest Neighbour	0.00	0.21	0.06	0.03
320.701				
	Best	Worst	Mean	Median
Random Forest	0.00	0.18	0.06	0.04
CART Tree	0.00	0.23	0.07	0.05
Adaboost.M1	0.01	0.20	0.07	0.05
Naive Bayes	0.01	0.43	0.15	0.11
k Nearest Neighbour	0.01	0.16	0.07	0.07
302.66				
	Best	Worst	Mean	Median
Random Forest	0.00	0.51	0.17	0.14
CART Tree	0.00	0.47	0.15	0.14
Adaboost.M1	0.00	0.51	0.18	0.18
Naive Bayes	0.01	0.72	0.39	0.52
k Nearest Neighbour	0.00	0.24	0.11	0.10

Table 6.5: Prediction of declarable amount from activity $1 \dots n$

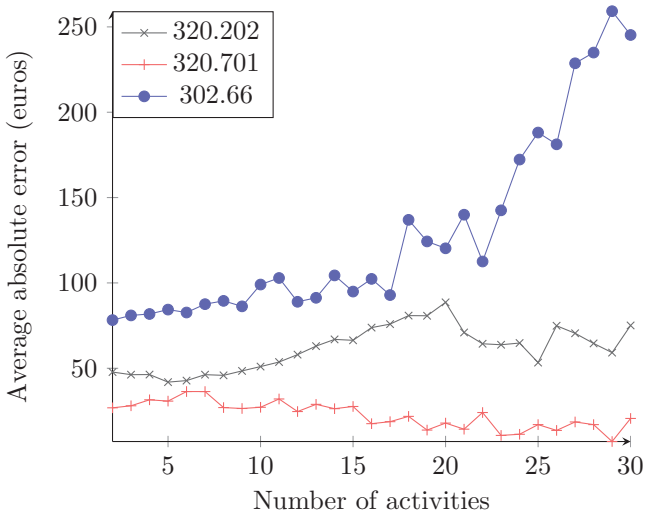


Figure 6.4: Random Forest declarable amount error for number of activities $1 \dots n$

column for that activity will have count zero. Since the 320.202 dataset is most accurately classified and has the lowest error rate for declarable amount, the initial best-case scenario uses this data set.

What this test shows is that the duration of care can be predicted in the best-case scenario with a standard deviation of 85 days. The average compound error (the same measure as for declarable amount) is one day. This means that we can say, with 95% confidence, that the care path will be finished within 190 days before or 190 days after the actual duration. This shows that duration prediction does not produce a usable result, given that a sub path always finishes within 365 days of its start date. Note that the standard deviation for the duration in the original data is 140 days, making the classified result an improvement of the most naive approach.

Whereas for the declarable amount the law of large numbers results in a usable result, here, it does not. The goal is not to know the total number of days the open sub paths will take, but it is to know when individual sub paths are finished, as this reveals when a cashflow is to be expected. The uselessness of this result means that test 3a (predict duration from the finished path) – which is essentially the same– will not be discussed further.

6.4.2 Predict activity n+1 from activity 1...n

Predicting the next activity is a possible approach to establish the whole sub path, which leads to the care product. Generally, this section shows if predicting the $n+1$ activity from the first n activities is a feasible approach. The data set is the same as for the first tests. Table 6.6 shows the results for predicting the n+1^{th} activity. The accuracy with which this can be predicted is significantly lower than for predicting the care product. This is likely caused by the greater number of unique activities compared to the number of unique care products. Also, the noise in the data when it comes to activity ordering has a negative effect on the accuracy of the classifiers.

Figure 6.5 plots the accuracy of the Random Forest classifier as a function of the number of activities in the sub path. The plot and the table both show that the accuracy of finding the next activity is low, the plot also shows that the accuracy becomes worse with more activities considered, but flattens out after 12 activities for the 302.66 and 320.202 data sets. 320.701 has a more erratic plot, but also has a low accuracy throughout. The trouble is obviously that an average accuracy of 0.3 over multiple activities possibly results in exponentially poorer accuracy: for 10 activities, the accuracy would be $0.3^{10} = 5.9e^{-6}$. Obviously, that is not a usable figure.

6.4.3 Predict activity n + 1...care product from activity n.

The previous tests have all involved classifiers to find a variable related to the number of activities. This test involves a shortest path algorithm to find the path in between an activity and a care product. This requires the care product as an intermediate step. To do this, we use the most accurate classifier we have found: Random Forests. There are two algorithms we test: Dijkstra's and Floyd-Warshall's shortest path algorithm. We use the classification from the first 6 activities to test the two algorithms, as this was shown to be the most accurate sub set for classification. The shortest path algorithms are trained/run

320.202				
	Min	Max	Mean	Median
Random Forest	0.15	0.40	0.25	0.24
Decision Tree	0.13	0.33	0.21	0.20
Adaboost.M1	0.08	0.38	0.23	0.24
Naive Bayes	0.09	0.30	0.18	0.19
k Nearest Neighbour	0.08	0.43	0.18	0.15
320.701				
	Min	Max	Mean	Median
Random Forest	0.00	0.43	0.18	0.16
Decision Tree	0.00	0.28	0.11	0.10
Adaboost.M1	0.00	0.30	0.17	0.17
Naive Bayes	0.00	0.31	0.18	0.20
k Nearest Neighbour	0.00	0.75	0.14	0.11
302.66				
	Min	Max	Mean	Median
Random Forest	0.16	0.42	0.26	0.25
Decision Tree	0.11	0.48	0.23	0.21
Adaboost.M1	0.12	0.47	0.28	0.26
Naive Bayes	0.09	0.33	0.22	0.22
k Nearest Neighbour	0.08	0.47	0.18	0.14

Table 6.6: Prediction of activity n from activity $1 \dots n$

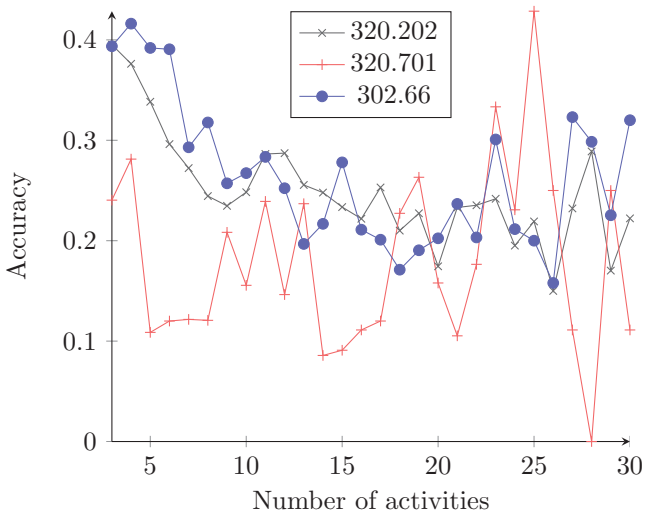


Figure 6.5: Random Forest accuracy of activity +1 for number of activities $1 \dots n$

on a 95% sub set of the available sub paths, to make sure that the graph is maximally complete. If the graph would omit certain activities or care products, there could not be a prediction. This noise in the training set is prevented by the large set of sub paths.

Obvious from the results is that Floyd-Warshall is more accurate than Dijkstra for this test. The output of the test is calculated as the number of correct activities in the prediction as a fraction of the number of actual activities. Interestingly, the 320.202 has up till now shown similar accuracy results for any algorithm compared to 320.701. It would appear that the size of the first data set leads to poorer results.

Accuracy as a function of the number of activities considered is shown in figure 6.6. The plot shows that the accuracy of the prediction of the path from activity n to care product is roughly equally accurate for $n \in \{1 \ldots 30\}$. For completeness sake, figure 6.7 shows the number of rows available as a function of the activity number.

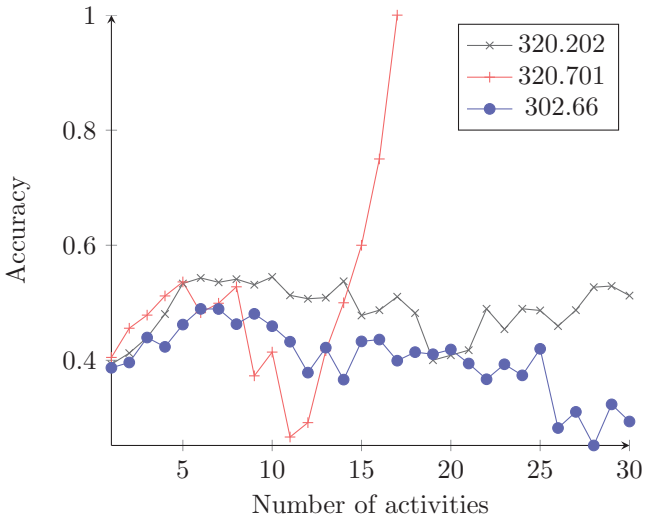


Figure 6.6: Average accuracy of path between activity n and care product for activity $1 \ldots n$

6.4.4 Predict activity n+1...final activity from activity n

This test is about using the shortest path algorithms as a prediction of the care product. The purpose of the test is to see if this approach might be more accurate than the classifier approach described in the first test. Table 6.8 shows the results for this test. What the table shows is that this method of care product prediction is significantly less accurate than the (worst) classifier-methods described earlier. Also notable is that –again– the performance is worst for the largest data set: 320.202.

320.202				
	Min	Max	Mean	Median
Floyd-Warshall	0.39	0.54	0.49	0.49
Dijkstra	0.06	0.09	0.07	0.07
320.701				
	Min	Max	Mean	Median
Floyd-Warshall	0.27	1.00	0.48	0.48
Dijkstra	0.11	0.22	0.15	0.16
302.66				
	Min	Max	Mean	Median
Floyd-Warshall	0.25	0.49	0.40	0.41
Dijkstra	0.10	0.22	0.15	0.13

Table 6.7: Prediction of path between activity n and care product

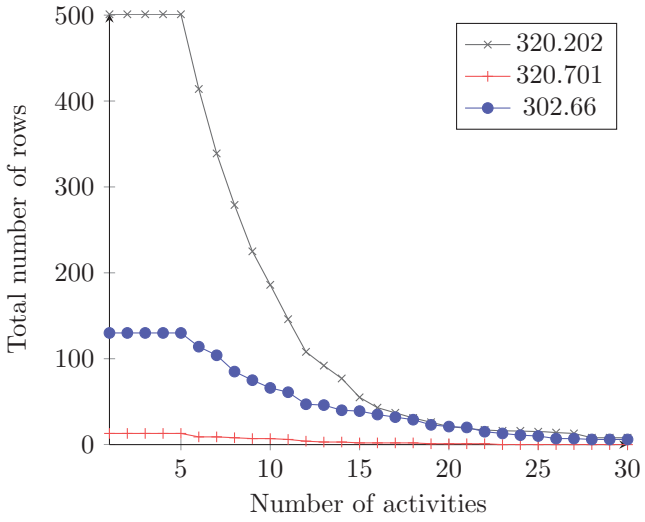


Figure 6.7: Number of rows for number of activities $1 \dots n$

6.4.5 Hypothesis conclusions

Given the results described in the previous sections, this section sums up what impact they have on the hypotheses. For every hypothesis, we will briefly discuss whether it should be rejected or not.

Hypothesis 1 states that the care product can be predicted with greater accuracy than $\frac{1}{|\{care\ product\}|}$. For 320.202, 320.701 and 302.66, respectively, the number of unique care products is 22, 6 and 11. For data set 320.202, the average accuracy of the best classifier is 0.47. For 320.701, it's 0.55. For 302.66, it is 0.40. These are greater than $\frac{1}{22}$, $\frac{1}{6}$ and $\frac{1}{11}$, so we *will not reject* this hypothesis.

Hypothesis 1a. states that the care product can be predicted with greater accuracy than $\frac{1}{|most\ frequent\ care\ product|}$ (predicting it will be the most frequent care product). Two out of three datasets are above this threshold for every tested number of activities, as is seen in figure 6.2. The third dataset is below the threshold for eight out of thirty tests. This is reason to conclude that this hypothesis *will not be rejected*.

Hypothesis 2a. states that prediction of the care product gets better with more activities performed. As figure 6.2 shows an increase in accuracy until around activity 6, and then drops for two of the data sets, the hypothesis should not be accepted. Figure 6.3 does show a steady increase in accuracy for one of three datasets, and a net increase between activities $1 \ldots 2$ and activities $1 \ldots 10$. This is reason to not reject this hypothesis, but it requires further study (with perhaps more or larger datasets) to give a definitive conclusion. Therefore, we mark this hypothesis as *inconclusive*.

Hypothesis 2b. states that declarable amount prediction gets better with more activities performed. Two out of three data set show an increase in error with more activities performed. Although there is –again– a possible correspondence with the decrease in number of rows, there is an increase from the first activity onwards. Therefore, this hypothesis is *rejected*.

Hypothesis 2c. states that duration prediction gets better with more activities performed. As we have concluded that duration prediction is not feasible for the existing data sets, this hypothesis remains *inconclusive*.

Hypothesis 3 states that it is possible to predict the duration with a smaller standard deviation of duration error than the standard deviation of duration error in the data set. For data set 320.202, we have shown that the standard deviation of the errors in duration prediction is 85 days for the classifier, versus 130 for the data set. Therefore, this hypothesis *will not be rejected*.

Figure 7.1 shows the whole process from business question to result graphically.

320.202				
	Min	Max	Mean	Median
Floyd-Warshall	0.00	0.06	0.02	0.02
Dijkstra	0.00	0.00	0.00	0.00
320.701				
	Min	Max	Mean	Median
Floyd-Warshall	0.00	0.69	0.30	0.33
Dijkstra	0.00	0.46	0.11	0.00
302.66				
	Min	Max	Mean	Median
Floyd-Warshall	0.00	0.58	0.24	0.10
Dijkstra	0.00	0.28	0.09	0.03

Table 6.8: Prediction of care product using shortest-path algorithms

Chapter 7

Framework

This chapter describes how the results presented in the previous chapter are translated in a general approach for answering process-related business questions. The research question this section addresses is the fourth and final question: What are the processing steps towards answering a business question? The steps induced from the performed experiment are described as they would appear in sequence below.

7.1 Translating the business question

The first step to answering a business question is assessing what the question is. Is it about a process related variable, or about predicting the path through a process graph? If it is about neither, then this framework is not suitable. Of course, there might be multiple business questions, such as in the hospital case.

Besides deciding what type the question is, the framework user should set in advance what the requirements on the answer are and how these should be made operational. The accuracy or completeness required of the answer will differ from question to question, and the accuracy and/or completeness measure will as well. For example, accuracy for predicting classes is the number of correctly classified instances as a fraction of the total number of instances –although different measures might be possible, see the discussion in the previous chapter on 'wrongness' of classification. Accuracy in predicting the path between two nodes on a graph is an entirely different measure, which depends on what we want to answer.

7.2 Data retrieval

Regardless of the type of question being answered, the techniques described in the previous experiment all have similar data requirements in order to work properly. To populate a process graph, the steps in the process must have distinct labels, and the sequence of the steps should be clear through every subsequent step having a unique date. Every instance of the process –equivalent with a path through the process graph– must have a value for the variable we

want to predict, if there is such a variable. Also, the instance must have a unique identifier, to distinguish it from the others.

7.2.1 Data analysis

Given these requirements, an inventory should be made of the available data. Does the data meet the requirements, and is there enough data to perform meaningful analysis? The latter is difficult to answer, but given the techniques described in this research, the more data, the better. If there is no data, or the data does not fit the requirements, the question answering process ends.

An important aspect of data analysis is to determine the amount of noise in the data. This means looking at the variables related to process instances and at the sequence of process steps, especially. The hospital case has a noisy data set when it comes to the sequence of process steps: it is more or less random to some extent. In the best case, this reduces the accuracy of the naive graph elicitation technique, in the worst case this kind of noise reduces the classification and path prediction accuracy.

7.2.2 Data transformation

If the data is complete in respect to the requirements, the data must be transformed to suit the techniques. Both the classification and process elicitation technique need data to be grouped in cases or instances, with cases modeled as a row of data. The variables belonging to the case are the columns in the row. An example of a case is a sub path in the hospital case, that holds the variables of interest for classification and prediction, as well as the process steps.

For the techniques described in this report, the format of the data row must be of the form *id,*{*variable 1, . . . , variable n*}, {*activity 1, . . . , activity k*}. The number of variables must be the same for every row, but the number of activities may vary.

7.3 Classifier training

If the business question is about variable prediction, the next step is to train the classifier. As the results have shown Random Forests to be the best performing classifier, this is the algorithm of choice. Training the classifier means taking a sub set of the available, transformed, data, and runnning the classifier on that data. The training data should contain the variable to predict, the rest of the data (obviously) should not. This imposes the restriction on the framework that it can only be used in a scenario where the variable to predict is known for some combination of activities, a general limitation of data mining.

7.4 Process graph elicitation

If the business question is about finding paths, a process graph is elicitated for the next step. The naive process elicitation technique used in this research creates a directed edge for any set of two subsequent steps in the data. Furthermore, it assigns a weight to each edge, depending on how often an edge occurs.

If the data has outcomes for each process instance (such as the care products in the hospital case), these are added as the final nodes in the graph. If the data consists of multiple sets, multiple graphs are made. A single start node is added for each graph, to provide shortest-path algorithms with a single starting point for the graph.

7.5 Prediction of variable

Variable prediction is done by taking the trained classifier and supplying it with data. The classifier then outputs variables for that data set. These variables could be classes or values, such as an amount or duration. The variables are finally interpreted, to provide an actual answer to the business question.

7.6 Prediction of path

Prediction of the path between a node and another node requires a "trained" shortest-path algorithm. The algorithm used is Floyd-Warshall, as this has been shown to be the most accurate for finding the path between nodes. Training the shortest-path algorithm means running the algorithm on the data set. In the case of Floyd-Warshall, the result is a matrix of distances between nodes. This is then used to give the path with the highest distance, where distance is the total weight of the edges between nodes. This weight represents the number of times the edges are traversed in the data, so the highest weight means the highest probability. As the path can only be predicted between two nodes, one node might need to be predicted itself, using the Random Forest classifier.

7.7 Reviewing results

The final step of the process is to compare the results from the analysis (prediction of variables or paths) to the requirements set on those answers. This means that the classifications and predictions are compared to the actual values for some test set, to see how accurate those results are. If the results are less accurate than is the requirement, the business question cannot be answered reliable. If the results are accurate enough, they can be interpreted. For example, in the hospital case, the Random Forest classifier can be used to determine the total declarable amount for open sub paths (this has around 10% inaccuracy), but the results for duration mean that question remains unanswered.

Figure 7.1 shows the whole process from business question to result graphically.

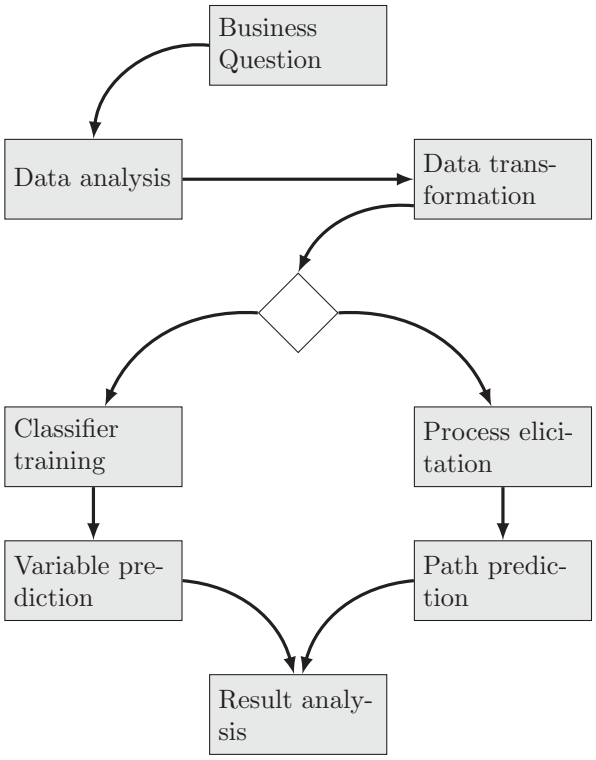


Figure 7.1: The framework process for answering a process related business question

Chapter 8

Framework Validation

This chapter describes a different case, used to validate the framework steps as described in the previous section. The new case involves a bug-tracking system called Mantis. Below, the steps from the framework are applied to this case. First, we will briefly describe the case.

Mantis is used by Topicus to assign bugs to individuals within the organization and to record the progress of those bugs. A bug always has a status, a closed bug has a resolution. Possible resolutions are "resolved", "will not fix", "not a bug" and so on.

8.1 Translating the business question

As there is no business problem underlying the Mantis case, a relevant question has to be found. For bug-tracking, it would be nice to know what the likely resolution to a bug will be. If it is "will not fix" or "not a bug", work on that bug might as well stop. Also, we might want to know what the statuses are the bug will go through until is finished, as an indication of the amount of time and effort that will go into the bug.

This makes the business questions "Predict the resolution of a bug" and "Predict the following statuses for a bug". As around 60% of the bugs ends up in the resolution "resolved", we will take that as the minimal accuracy of the classifier for the first question. As there is no apparent threshold for the second question, we will choose 75% as the minimal accuracy.

8.2 Data analysis

The data available for the Mantis case consists of two database tables: a "Bug" table that has a bug id, an id for project that the bug belongs to, an id for the person reporting the bug, a priority and several other fields. The other table – called "Bug history"– holds the field changes for a bug, and records the identifier for the bug being changed as well as the old and new value for the field and the date the field was changed. The field of interest is the status field. The database columns of interest are the bug id, the resolution, the statuses and the dates they were changed.

8.3 Data transformation

To get the data in the format required to perform the transformation (which is implemented in Java), a query was performed on the Mantis database that joins the Bug and Bug History tables. This results in a set of tuples with columns *id, resolution, status, date*. There are 2450 rows in the data base. Once transformed, the data is in the required general format for variable and path prediction:

$$id,\{variable\ 1,\ \ldots,\ variable\ n\},\{activity\ 1,\ \ldots,\ activity\ k\}$$

For the Mantis case the format is *id,resolution,*{*status 1, . . . , status n*}, as resolution is the variable of interest per the business question. The transformed data has 845 rows, the average number of successive statuses per bug is two, the maximum number is 12.

8.4 Classifier training and variable prediction

This step involves taking a sub set of the available data –that has a value for the resolution– and using it to train the Random Forest. Next, another sub set of the available data is used to test the classifier. The test results tell how useful the answer produced is. The question that the classifier test answers is the first question: predict the resolution of a bug.

8.5 Process elicitation and prediction of path

Prediction of the next statuses for a bug is a little more complicated than the classifier training, as this requires to induce a graph on the statuses and to determine the shortest paths using Floyd-Warshall. As we want to predict the path to the resolution, knowing the resolution is a prerequisite as well: Floyd-Warshall determines the distance between two nodes. This means using the classifier training for this step as well. The classifications for the number of statuses that is the most accurate is the set of classifications used. The set of paths (sets of statuses) produced when the graph algorithm/Floyd-Warshall/ Random Forest combination is run (on a sub set of the data) is compared to the actual following statuses to determine the accuracy of path prediction.

8.6 Reviewing results

The accuracy of the classifier for multiple runs –on length of statuses 1,2 and so on– and for the graph algorithm are shown in table 8.1. The accuracy measures used are for variable prediction:the ratio of correctly predicted variables to the total amount of tuples, and for path prediction: the number of correctly predicted path elements as a fraction of the total number of actual following path elements. Figure 8.1 shows a plot of the accuracy for different numbers of statuses included, figure 8.2 shows the total number of rows per number of statuses.

Mantis dataset				
	Min	Max	Mean	Median
Variable prediction	0.33	0.72	0.57	0.63
Path prediction	0.81	1.00	0.89	0.87

Table 8.1: Results for the Mantis case

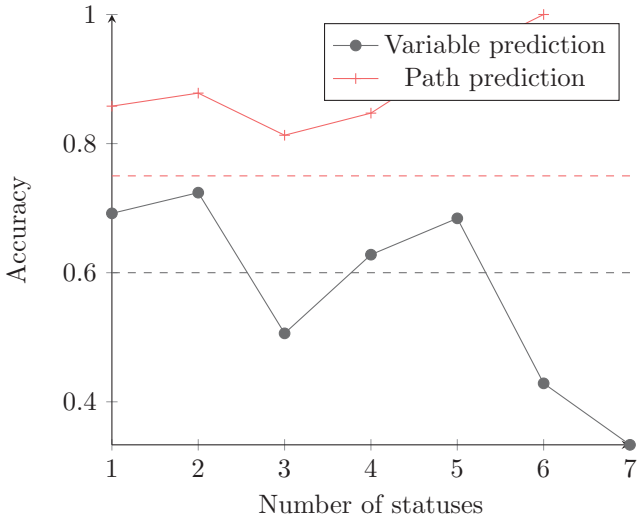


Figure 8.1: Average accuracy as a function of the number of statuses

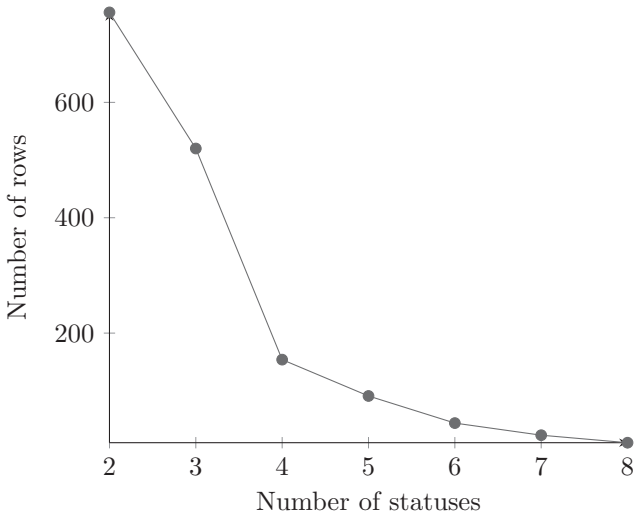


Figure 8.2: Number of rows per number of statuses

On average, the variable prediction performs just under the threshold, but in four out of seven cases, it actually performs better. This is shown in figure 8.1 as the black dots above the dotted black line. Similarly, path prediction is better both on average and in the worst case scenario than the threshold, here respresented by a dotted red line. The accuracy of the variable prediction is worst for the number of statuses that has only few rows. This leads us to suggest that the trained Random Forest is a good enough predictor of the resolution to answer the business question. The only thing to take into account is the number of statuses, if it is greater than two, accuracy becomes more erratic. For path prediction, the results are more positive: for any number of statuses, the prediction for the path to resolution is accurate enough to be useful.

Chapter 9

Conclusions

There are four questions and one class of problems this research has aimed to answer. This section describes the extent to which it has succeeded in reaching these goals. We will start by discussing the answers to the research questions. Second, we will reflect on the research and business problem presented at the start of this research. Finally, we will discuss future work that builds on the work presented in this research.

9.1 Methods for finding a process graph

The research has shown that finding a process graph, or process elicitation, there are two approaches: Choose a process mining algorithm such as the α-algorithm, or use a more naive approach. As the latter can deal with loops in the process, this is the approach of choice. Combined with the Floyd-Warshall algorithm to find the most likely path between nodes, this approach deals with both loops and noise. Furthermore, in terms of the completeness measure defined in chapter 3, the naive approach performs better than the other algorithms, as it simply includes every edge found in the data. Since Dijkstra will not be used, the naive elicitation does not relabel duplicate nodes within a trace (see the discussion in chapter 5.2.2).

9.2 Methods for finding the subsequent path

The results in chapter 6 and 8 have shown that the Floyd-Warshall algorithm combined with a prediction of the end node leads to reasonable results for data sets with little noise. For a noisy data set, such as the hospital case, where the sequence of activities in the data is implicit or does not match the reality, the accuracy of Floyd-Warshall/Random Forests is below 30%. For the less noisy Mantis-dataset, the accuracy is much higher: around 80% on average. The other approach tried is Dijkstra to do similar prediction: the accuracy for this approach is below 10% on average, for any of the data sets tried.

9.3 Methods for finding path-variable correlations

The literature research has identified Random Forests as the best technique for finding a path-variable relation, followed in accuracy by boosting, neural networks and bagging. This has been tested in the hospital case, which has shown that Random Forests is indeed consistently the best performing classifier. For predicting a directly path related variable, such as the care product or declarable amount in the hospital case, the accuracy is highest. For a variable that is not the consequence of the steps in the path, such as duration, the accuracy is poor. Even in this worst-case scenario, the error of the prediction is lower than the error in the data set.

9.4 Process steps for answering a process-related business question

Chapter 7 has identified eight steps for answering a business question, as derived from the approach taken in the hospital case experiment. These steps share many characteristics with a generic data-centered approach to a problem: collect data, analyze it, and review the results. The framework described, and as implemented in R and Java, shows in detail what every step involves and presents the algorithms/techniques to be used. That these steps work, and produce a result, has been shown in the Mantis case (chapter 8), which serves as validation.

9.5 Research problem

The steps in the framework, the selection of the techniques and the implementation of the framework combined form a solution for the research problem. They represent a generic way of finding the business question and consequently finding an answer to this question. The framework is the result of the work in chapter 6 and was validated in chapter 8. The framework has been shown to be a solution for the type of business question as presented in the validation (that are the consequence of the path), but there has not been a satisfactory answer to one question, that is not the direct consequence of the steps in the path.

9.6 Business problem

For the business problem to be solved, there has to be a way to predict the declarable amount and the duration of hospital sub paths. The results of the experiment have provided an answer for these two sub problems, but only the declarable amount can be accurately predicted. There is no threshold set for the accuracy of the declarable amount, but the average error between 7 and 17% seems like an acceptable accuracy. It is important to consider that the average error decreases with more data, a result of the law of large numbers. Predicting the duration, even in the best case scenario where the entire path and the care product are known leads to a prediction that has a standard deviation for its error of 85 days. This means that the result is not usable, even though it is

better than the standard deviation of the actual data. Therefore, we can say that the research has solved the problem it has set out to solve only partially.

9.7 Future work

The fact that the research has only partially been able to solve the business problem leads to one element for future research: Predicting variables that are not the consequence of path variables. As the results of this research seem to function poorly for this task, future work will have to show if other approaches fair better. Another element for future work is to test the framework in practice, and to keep improving it. It is after all well possible that new algorithms and techniques emerge that are better than those presented in this work.

Another element for future study is the second hypothesis in the Hospital Case chapter: does accuracy increase with more nodes included? So far, the results have been inconclusive. It will be interesting to see if this hypothesis is a general property of processes, or that it should be rejected.

Finally, visualizing the process graph and analyzing the result could be interesting, to see how the make up of the graph influences the accuracy of the prediction. For example, a graph with low complexity could lead to better prediction of the path (fewer choices make it easier). Whether or not this is the case, and what other interesting aspects of the elicitated graph are, is something for future research.

Bibliography

[1] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. *Advances in Database Technology–EDBT'98*, pages 467–483, 1998.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, 1993.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo, et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12:307–328, 1996.

[4] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499. Citeseer, 1994.

[5] E. Alfaro, N. García, M. Gámez, and D. Elizondo. Bankruptcy forecasting: An empirical comparison of adaboost and neural networks. *Decision Support Systems*, 45(1):110–122, 2008.

[6] Algowiki.net. An implementation of the Floyd-Warshall algorithm in Java, 2011. Visited on November 1st, 2011.

[7] E. Alpaydın. *Introduction to Machine Learning*. The MIT Press, 2004.

[8] S. Baase and A.V. Gelder. *Computer algorithms: introduction to design and analysis*. Addison-Wesley, 2000.

[9] R.E. Banfield, L.O. Hall, K.W. Bowyer, and W.P. Kegelmeyer. A comparison of decision tree ensemble creation techniques. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):173–180, 2007.

[10] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[11] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[12] O. Burkart and B. Steffen. Model checking for context-free processes. In *CONCUR'92*, pages 123–137. Springer, 1992.

[13] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20:273–297, 1995.

[14] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[15] S.P. Curram and J. Mingers. Neural networks, decision tree induction and discriminant analysis: an empirical comparison. *Journal of the Operational Research Society*, pages 440–450, 1994.

[16] J. Desel and W. Reisig. Place/transition petri nets. *Lectures on Petri Nets I: Basic Models*, pages 122–173, 1998.

[17] Y. Freund and R.E. Schapire. A desicion-theoretic *(sic)* generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[18] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithms. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[19] P.O. Gislason, J.A. Benediktsson, and J.R. Sveinsson. Random forests for land cover classification. *Pattern Recognition Letters*, 27(4):294–300, 2006.

[20] G. Greco, A. Guzzo, G. Manco, and D. Saccà. Mining and reasoning on workflows. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):519–535, 2005.

[21] P. Green and M. Rosemann. Integrated process modeling: an ontological evaluation. *Information systems*, 25(2):73–87, 2000.

[22] C.W. Gunther, S. Rinderle-Ma, M. Reichert, and W.M.P. Van Der Aalst. Using process mining to learn from process changes in evolutionary systems. *International Journal of Business Process Integration and Management*, 3(1):61–78, 2008.

[23] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Morgan Kaufmann Pub, 2011.

[24] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

[25] D.J. Hand. Data mining: Statistics and more? *The American Statistician*, 52(2):112–118, 1998.

[26] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[27] J. Herbst and D. Karagiannis. Workflow mining with InWoLve. *Computers in Industry*, 53:245–264, 2004.

[28] B. Kiepuszewski, A.H.M. ter Hofstede, and W.M.P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39:143–209, 2003.

[29] S. Kotsiantis and D. Kanellopoulos. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.

[30] D. Meyer, F. Leish, and K. Hornik. The support vector machine under test. *Neurocomputing*, 55:169–186, 2003.

[31] 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. ACM, 1995.

[32] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1, 1986.

[33] J.R. Quinlan. Improved use of continuous attributes. *Journal of Artificial Intelligence Research*, 4, 1996.

[34] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.

[35] A. Savasere, E. Omiccinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 432–444. Citeseer, 1995.

[36] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2006.

[37] W.M.P. van der Aalst and C.W. Günther. Finding structure in unstructured processes: The case for process mining. In *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on*, pages 3–12. IEEE, 2007.

[38] W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business process mining: An industrial application. *Information Systems*, 32:713–732, 2007.

[39] W.M.P. van der Aalst, V. Rubin, H.M.W. Verbeek, BF van Dongen, E. Kindler, and C.W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.

[40] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: a survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.

[41] W.M.P. van der Aalst and A. Weijters. Process mining. *Process-Aware Information Systems*, pages 235–255, 2011.

[42] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1143, 2004.

[43] B.F. van Dongen, A.K.A. de Medeiros, HMW Verbeek, A. Weijters, and W.M.P. van der Aalst. The prom framework: A new era in process mining tool support. *Applications and Theory of Petri Nets 2005*, pages 444–454, 2005.

[44] A. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer Aided Engineering*, 10(2):151–162, 2003.

[45] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

This document was typeset using LaTeX. Images were created using the Ti*k*Z package.

Appendix A

Typical DOT Care Process

Using the DOT system, the (administrative) process for the hospital starts when a patient reports a medical complaint to a specialist, which normally would happen at a consultation. The specialists opens a care path. Alternatively, the patient could be redirected from the ER. In that case, the ER will have opened the care path. In general, the care path is opened at the first medical treatment or assessment someone undergoes while in the hospital.

The person opening the care path (usually a specialist known as the *gate specialist*) sets the diagnosis and starts a sub path with a specified care type. The care type specifies the general category of care the patient is to receive. The care types are listed in table A.1.

#	*Care type*	*Description*
11	**Initial treatment**	This is the standard care type for the start of medical treatment.
13	**Inter-collegial consultation**	This is used if the patient has been referred by another specialist ("second opinion").
21	**Treatment continuation**	for a continuation of care for the same diagnosis.
41	**Alternative product**	for delivering an alternative product or for care provided by non-DOT specialisms.
51	**Intensive care with consultation**	The care type for a patient who needs intensive care as well as a consultation with a specialist.
52	**Intensive care without consultation**	The care type for a patient who needs intensive care but does not need a consultation with a specialist.

Table A.1: Care types in the DOT system

Of these care types, type 11 and 21 are the most common: they are used for patients coming in for a clinical consultation, who constitute the majority of

people using the hospital's services. Types 13 and 41 are variations on 11 and 21 that describe a different starting point for the patient (for type 13: another specialist redirecting the patient, for type 41: a patient requiring non-standard care or care not covered by the DOT system, such as dental surgery). Types 51 and 52 take a different starting point altogether, as they represent a patient that is brought to the intensive care unit instead of starting with a consultation. The overall care path exists of at least one sub path with one of the six mentioned care types. An additional rule exists as to when care type 52 can be used: only when there are no applicable type 11 or 21 sub paths to be taken, a sub path with type 52 can be used. This makes sense when looking at the definition: it is for intensive care (hospitalization) without regular care (in other words, a consultation). Corollary, type 51 is only used in combination with a type 11 or type 21 sub path.

Sub paths consist of the actual care the patient is to receive. A sub path consists of zero or more activities (such as treatment sessions, examinations or lab work) and medications (or other materials). The chosen sub path depends on the patient's diagnosis. The sub path functions as a guide for the specialist, to show what the standard treatment for a certain diagnosis is. If the actual care the patient ends up receiving differs from this standard care, then this will be taken into account upon categorizing the provided care as a DOT.

Closure rules are used to decide when a sub path is finished. Once a sub path is closed, the hospital can declare it at the insurance provider (after categorization as a DOT). There are different rules to decide when a sub path is to be marked closed, depending on the care type. Table A.2 shows the rules per care type. Note the difference between care path and sub path: the latter is part of the first.

Care type	*Closure rule*
11/21	The sub path is marked closed after 42 days since the last activity have passed, if the patient was treated clinically(admitted) or was operated polyclinically. If the patient was not operated, the sub path is marked closed after 90 days for type 11, 365 days for type 21. The overall care path is closed 365 days after the last sub path was closed, or after the patient has passed away.
13	The care path (and therefore any sub path) is closed after the patient is discharged from the hospital.
41	The care path is closed after the last activity was performed.
51/52	The care path is closed after the patient is discharged from the hospital. For type 51, it is also required that the related type 11/21 sub path was closed.

Table A.2: Closure rules for different care types

Example Figure A.1 shows an example care path, consisting of three sub paths. The first sub path is (as it is normally) of type 11, indicating it is the start of

the care path. The following sub path cannot have the same type, therefore it has type 21 (Continuation), as it follows the first sub path. This is not a rule, so this path could have any care type, except 11 (there already is a first sub path) and 52 (there is a path with type 11). The final path shown represents an admittance in the Intensive Care unit, which has to be denoted by type 51 (because type 52 is not allowed).

After each sub path, there is a period of no activity, denoted by a diamond. As mentioned before, there must be 42 days without activity before a type 11 sub path can be closed. The same goes for the second sub path (with type 21). In conclusion: the figure shows two closed sub paths and one open sub path, as there could be more activities in sub path 3. The overall care path is therefore open as well.

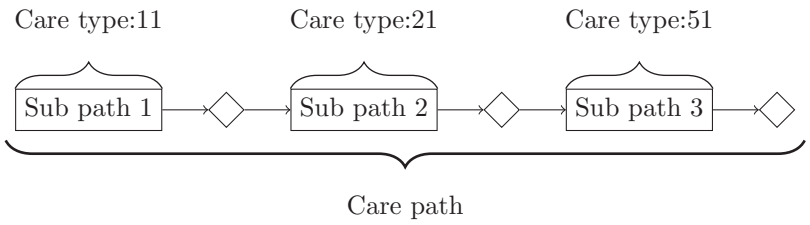


Figure A.1: An example care path, with three sub paths