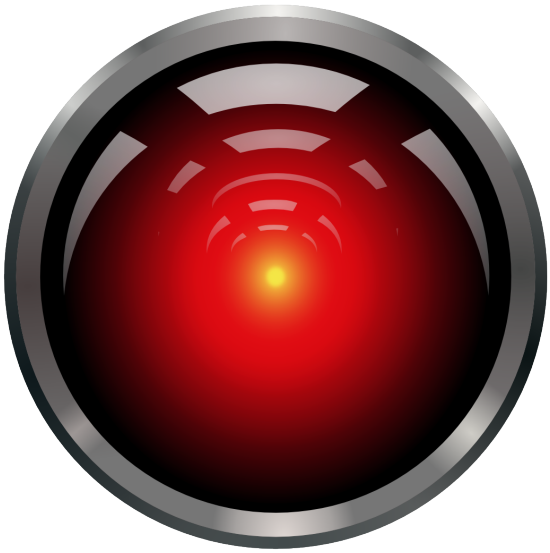
CHALMERS



Automating Operational Business Decisions Using Artificial Intelligence: an Industrial Case Study

Master's thesis in Software Engineering and Technology

PIER JANSSEN
MACIEJ WICHROWSKI

Department of Computer Science and Engineering
Division of Software Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2012

MASTER'S THESIS IN SOFTWARE ENGINEERING AND TECHNOLOGY

Automating Operational Business Decisions Using Artificial Intelligence: an Industrial Case Study

PIER JANSSEN
MACIEJ WICHROWSKI

Department of Computer Science and Engineering
Division of Software Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2012

Automating Operational Business Decisions Using Artificial Intelligence: an Industrial Case Study
PIER JANSSEN
MACIEJ WICHROWSKI

Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
HAL 9000 (Heuristically programmed ALgorithmic computer), from *2001: A Space Odyssey.*

Göteborg, Sweden 2012

Abstract

The process of making business decisions is increasingly reliant upon analyzing very large data-sets. Due to the amount of decisions having to be made on a daily basis, this becomes time-consuming and expensive to carry out manually.

The purpose of this thesis was to determine whether using Artificial Intelligence to automate business decisions is feasible. This was done by carrying out a proof of concept project at IFS World, a software company developing Enterprise Resource Planning systems.

Procurement decision making was chosen as a case for this study. Automating these decisions can not only result in speeding up the decision making process, but also in making more accurate decisions. To achieve this, three machine learning algorithms were proposed. Their goal was to learn preferences from historical procurement data and apply this knowledge to new situations. Prototyped versions of the algorithms were developed, tested and compared using both real-world and artificial datasets.

The results showed that after a short period of supervised learning, two algorithms were able to make decisions automatically, with a low error-rate. Furthermore, sensitivity analysis showed that the algorithms are robust enough to recover from errors in the training data. The study also revealed several constraints and prerequisites related to feature selection, data freshness, and completeness. It was concluded that automating operational business decisions using Artificial Intelligence is achievable if certain preconditions are met. It can provide several advantages over manual decision making: it will speed up the decision making process, and can, in certain scenarios, improve the quality of the decisions.

Keywords: artificial intelligence, machine learning, operational business decisions, procurement

Acknowledgements

First and foremost we would like to thank our academic supervisor, Dr. Lars Pareto, for his support throughout the entire process of creating this thesis. From finding a suitable subject, to writing the thesis itself, his ideas and feedback have been most valuable.

We are also grateful to IFS World for providing the opportunity to carry out this study at their facilities. In particular, we want to thank Dan Matthews, David Andersson and Mikael Hultin for taking time out of their busy schedules and providing input and feedback.

Finally, we would like to express our gratitude towards one of the industrial partners of IFS World, who supplied us with procurement data from their systems. This has given us a valuable insight into the procurement process and the parameters involved.

Contents

1 Introduction

Making business decisions nowadays is increasingly reliant upon analyzing very large data-sets and the complex relations between them. This makes the task time consuming and complex for humans to carry out accurately. Algorithms could support or even take over this task by learning to make certain decisions automatically as part of an Artificial Intelligence (AI) system.

The research area of AI is broad, with topics ranging from cognitive psychology to statistical analysis and theoretical mathematics. Many of these areas have been heavily researched since the 1960's. However, applying AI to business intelligence is a relatively unexplored area, and it is not yet clear whether current technologies are applicable to business decision problems.

Business decisions can be divided into various categories based on their level of abstraction, ranging from high-level, long term strategic decisions to less complex operational decisions made on a daily basis. The latter provides the best opportunity for automation, because of their repetitive nature and relatively small scope [29]. A single operational decision, for example procuring a part for a product to be assembled, may be perceived as insignificant on its own. However, because many of these decisions are made every day, their aggregate does become valuable. This thesis focuses primarily on automating these operational business decisions.

Making operational decisions requires processing large quantities of data, which most modern, large companies handle using an Enterprise Resource Planning (ERP) system. These systems provide functions for fast data retrieval, as well as presenting the data to the user in various forms, such as spreadsheets and diagrams. The main purpose of an ERP system is to enable companies to implement streamlined business processes. The business logic incorporated in the system is designed to provide support for automatically repeating certain decisions (e.g. always buying a part from a specific supplier), but most decisions need to be made manually.

There are many different approaches to AI in general. Well-known approaches are Intelligent Agents, case-based reasoning and neural networks. These approaches all have their pro's and cons, and some are more developed and sophisticated than others. In addition, some will be more suitable to making operational business decisions than others. The suitability also depends on the domain in which the business decision is made. For example: decisions concerning the financial sector would have different characteristics compared to decisions made in health care [20].

In this thesis, we investigate how business decision making can be automated with the use of AI in the context of ERP systems. Specifically, we are interested in the following questions:

- Which criteria make an operational business decision a good candidate for automation using AI?
- How can the relative importance of features contributing to an operational decision be expressed?
- With this relative importance, how can an operational business decision be made automatically?

We investigated these questions with a supply chain management system provided by IFS, a software company developing ERP systems. The test data for this case study was provided by one of their clients, a large size manufacturer.

2 Methodology

This thesis has been carried out following the proof of concept methodology. This involved the four main phases shown in figure 2.1. To begin with, we conducted a theory review to investigate the state of the art in the field of AI, including an exploratory evaluation of available technologies and frameworks. We also looked at decision making on a psychological level, to understand what drives the decision makers, and how it relates to AI. By looking at prominent AI publications such as [20, 10] we picked several candidates which expose different AI viewpoints. Approaches that seemed most promising were further evaluated by the creation of small mockups using existing

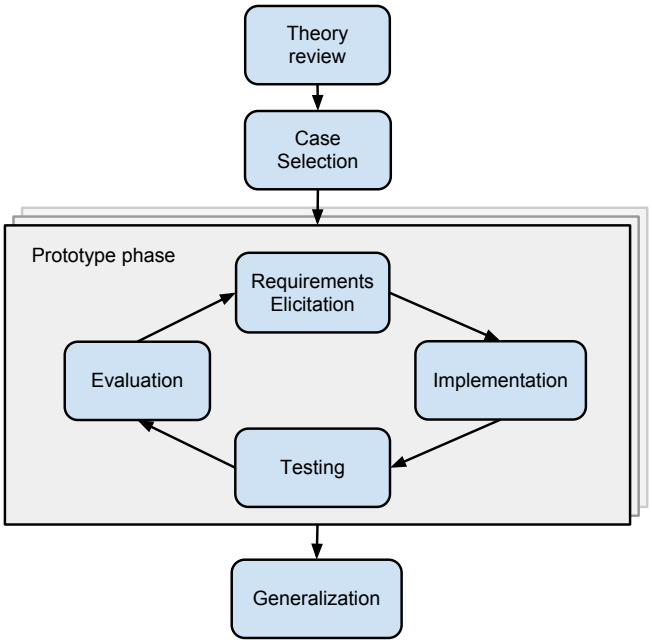


Figure 2.1: *Schematic view of the research process.*

frameworks. In addition to exploring the AI theory, we also looked into the suitability and data requirements for the automation of different types of business decisions.

In the case selection phase, an appropriate business decision scenario was selected. We conducted a series of focus group meetings with R&D department members and developers from various domains (manufacturing, fault reporting, etc). During these meetings, various AI approaches were presented and discussed, and ideas for specific business cases were brought up. This was followed by a brainstorming session with a group of early adopters representing a small set of customers involved in testing the IFS application suite. The goal of these sessions was to verify whether the ideas brought up during the focus group meetings had real-world relevance, as well as to uncover new cases. The selection of the candidate case for a prototype was done in the final focus group meeting, during which the customers' ideas were discussed with R&D department members. The ideas were presented together with what we had found to be the most suitable AI approaches. From these ideas, one business case was selected based on the following criteria: data availability within the company, the problem's complexity, the quantifiability of its influencing factors, the expected generalisability and real-world relevance.

After the case selection, the prototyping phase started. This phase consisted of several iterations over four steps. The first step was elicitation of high-level requirements and understanding the problem and its domain. This was done through a series of interviews with a domain expert, internal to the company. Afterwards we reviewed previously picked approaches to find the most suitable one for the case. It turned out that none of the approaches found exactly matched the situation and therefore needed to be adjusted. Throughout the iterations of the prototyping phase, a number of variations on the algorithm were implemented. Because obtaining test-data which exhibited the desired properties proved difficult, it took several iterations before a clear benchmark for the algorithms emerged. The testing step was performed using both artificial data as well as real-world data supplied by various customers. The last step in the iteration (the evaluation phase) was to evaluate the findings and discuss adjustments for the next iteration.

After the prototyping phase, the solution's potential for generalization, areas for improvement, and opportunities for further research were discussed in a focus group meeting with the R&D department.

3 Case description

Our study was conducted in collaboration with IFS Labs, the R&D department of IFS World. IFS is a large-size, global software development company specialising in ERP systems.

The goal of IFS Applications, their business application suite, is to automate tasks and processes of large companies, and to support business decision making. The product provides a complete suite of component-based ERP software as presented in fig. 3.1. The customer can select which components are required for their business.

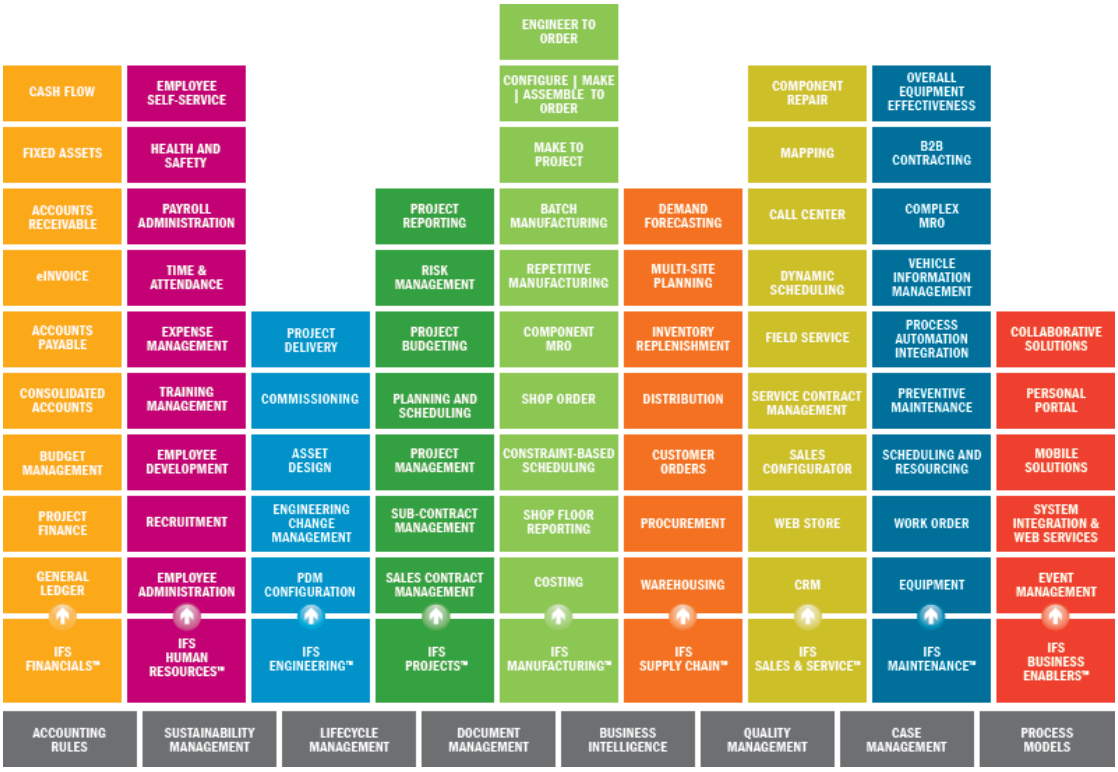


Figure 3.1: *An overview of the IFS Applications suite [7].*

The clients of IFS are mainly large, worldwide companies from many different domains such as manufacturing, construction, aviation and the automotive industry. They handle large volumes of data and have to make many operational decisions on a daily basis.

The IFS Applications suite allows users to create predefined decision paths and policies configured through parameter setup or process / work-flow maps in order to automate predictable decisions. However, these configurations are static, and require manual adjustment, making them both time-consuming and expensive to use. By introducing an AI solution with the ability to learn and adapt, this problem can be resolved.

The company runs an 'early adopters' program for their customers in which they can contribute to the late stages of product development. Some of them stated that the requisition process (picking the best supplier for a product purchase) is a typical case where a static system does not provide an ideal solution. The decision is frequently recurring and involves processing large volumes of data. There is a relatively limited number of features that influence the decision. Furthermore, these features can be easily represented as numerical values and are readily available within the application. The requisition scenario will be used as the case for this thesis.

3.1 Requisition process

The requisition process consists of the following steps that need to be looked into in order to get a full understanding of the reasoning behind the purchase decision.

The process starts by determining the demand for a specific part. This can be done using various methods, such as KanBan [31] or Material Requirements Planning (MRP) [30]. Some of these methods are fully automated for recurring part needs, while others are executed manually for incidental purchases. The result of this step is a purchase requisition which is entered into the system.

If a part is being bought for the first time, the purchase information (e.g. price, lead time) from the suppliers is not yet available. In order to get this information, a request for quotation is made. These are then returned by the suppliers, providing the purchase information.

For some parts the choice of supplier is determined by agreements between the supplier and purchaser. In this case, the purchase order is made automatically and does not require human interaction.

Usually the purchaser needs to decide from which supplier a part is going to be ordered. This is determined by comparing supplier's offers, and selecting the best matching candidate. This comparison is done based on the most desired quality (e.g. best price, shortest delivery time, etc.). The knowledge about the importance of the part's qualities is intuitive, rather than explicitly codified. However, the system offers a feature which allows the purchaser to predefine a "main supplier", which is the default choice for a specific part. Evaluating which supplier should be the main supplier is generally done yearly. Because comparing suppliers is time-consuming, purchasers tend to choose the main supplier without comparing alternatives. This may result in a sub-optimal choice, which exposes a company to unnecessary expenses.

Once the order is placed and the product has been received, further administration is performed. Among other things which are not directly relevant to this study, this results in a number of supplier quality measures that may be used in the previously described steps of the process. For example the recorded actual delivery time compared to the promised delivery time and the quality of the product (scrapped quantities).

3.2 Problem analysis

As described in the previous section, it is clear that the selection of the best-matching supplier can be improved. In particular the "main supplier" default choice can cause many sub-optimal decisions. To help purchasers make better decisions, a decision making algorithm could be introduced. This algorithm should 'learn' the purchaser's priorities for each part and use this to automatically suggest the optimal supplier choice.

Some of the early adopters raised some concerns about fully automating the supplier selection process. Due to this, the algorithm should first only make suggestions upon which the purchaser can make the actual decision. Once the purchaser trusts the algorithm enough, it can substitute the "main supplier" functionality.

Many parts exhibit the same requisition priorities, due to their similarity. For example, it is highly likely that resistors with a different resistance still have the exact same priorities. The system specifies "purchase commodity groups", which can be used to bundle these parts and make learning the priorities more accurate.

3.3 Available data

As part of the requisition process, the IFS application suite provides various data concerning placed orders. Each requisition consist of a requisition identification number and a part number. Similar parts are grouped by specifying a purchase commodity group identification number. Furthermore, all requisitions have latest order and wanted delivery data, which determine the deadline for part to be delivered. For each supplier that can deliver, there are a number of variables that are specific to that supplier, such as lead time, price etc. After a part is delivered it is possible to evaluate the supplier based on their punctuality and part's quality. Those features are the factors by which the suppliers are compared.

4 Theoretical framework

4.1 Decision making process

Over the past decades, much research has been done on the psychology of human decision making, and this has resulted in several decision making models. One of the first significant models was the Expected Utility model by Savage [24]. It considers a situation as a set of (external) events combined with a set of possible actions. For each combination, an expected utility value is defined. Given a set of probabilities for the events, the decision maker can use these values to choose the action that provides the highest utility. However, this model has received much criticism over the years. The main argument was that this model is unrealistic as it requires all events and their respective probabilities to be known beforehand, which is seldom the case. Furthermore, the number of combinations of actions and events grow exponentially when consecutive decisions have to be made, making it unfeasible to compute [26].

More recent models, such as recognition-primed decision (RPD) [12], take a different approach. In RPD, the decision maker attempts to pattern-match the given situation with previously encountered ones and tries to find the best matching solution. If a situation does not match exactly, the most similar one is chosen and the solution is adapted to the current situation. Then, the decision maker does a simulation in his mind to consider its suitability. This model has been refined and developed further by many researchers (Gilboa and Schmeidler [6], Schank [25] and Kolodner[13] among others), eventually resulting in Case-Based Reasoning, which we'll describe and discuss in a later chapter.

In a more generic sense, decision making can be expressed through value-driven thinking [11, 19]. First you examine your current state and define your goal, then you 'look ahead' and design actions to achieve this goal. One might also consider the influence of the decision maker's preferences and biases in choosing the action to take to reach a satisfactory future state. This gives rise to the concern of differentiating between actions and events. Performing an action may influence events, for example lowering the price of a product to sell will cause competitors to follow this. Furthermore, one has to define the boundaries of the model and consider the widening correlation between events and outcomes of actions as time progresses [3].

One major psychological aspect influencing decision making is cognitive biasing. Humans don't make decisions purely rationally, but are affected by irrational, emotional and external factors. For example people tend to take riskier actions in situations when they can lose. However in situations when they can profit, the same people prefer to stay conservative. Another example of biasing human perception is anchoring a point of view while making decisions. In this case the environmental factors and the human's previous experiences influence the outcome of the decision making process. For example, the great holidays at the Mediterranean sea from last year will shape your perception of next holidays at the Baltic sea. As a final example of biases, it is proven that most recent events are more relevant to making decision than those far from the past [1, 16].

4.2 Decision making and Artificial Intelligence

Applying the psychological models of human decision making to Artificial Intelligence has proved to be difficult for a number of reasons. First of all, computer based support systems require a high level representation of its surrounding environment to provide meaningful communication with the user. Without this, the system will not be able to do more than predefined mathematical computations. Transforming data (i.e. numbers and words without meaning) into information (i.e. data with relationships and a context for interpretation) requires the presence of a high-level information model. Capturing the data is relatively easy, whereas expressing the model and information is difficult and requires domain expert input [17].

Secondly, humans have the capability to make decisions based on a partial understanding of the environment [17]. On the other hand, AI systems don't have this capability and require a complete model of the world in which it operates. Defining this model completely is usually unachievable, due to the scale and undiscovered factors [18].

Thirdly, most factors that influence a decision have complex relationships, making it hard to identify the most relevant ones. The engineering principle of breaking a problem down into smaller sub-problems does not always work for complex decisions: the sum of the individual smaller decisions may not be equal to the larger decision into which they are combined [18].

Finally, a design decision has to be made about whether or not biases and other human factors should influence the decision making system. This depends on the purpose of the decision making system. If the system should mimic the human decision making process, then biases should be included. Other systems, such as expert systems, make decisions purely based on diagnosis and rationality. Their goal is to provide the optimal decision rather than to appear human-like. Therefore, biases are not relevant to these systems [18].

4.3 Artificial Intelligence approaches

For an approach to be suitable for a business decision support system, a form of heuristic ability is required. This will distinguish it from static pre-configured behaviour. It also means that before the system can be used, it has to be trained using a real-world data-set, or an accurate simulation environment. The benefit of this is that once it is in use, it continuously adjusts and balances itself, ensuring continuing accurate results.

While conducting the thesis we have looked at a number of promising AI paradigms [20, 13, 2]. This section will give a compact overview of the most prominent approaches that are relevant to this project: Linear regression, Neural Network, Case-Based Reasoning, and Intelligent Agents. Furthermore, the applicability of each approach will be related to the exemplar case.

4.3.1 Linear Regression and Gradient Descent

Regression analysis attempts to find a relationship between one or more variables X and outcome values y. Linear regression specifically targets linear relationships. Given a parameter vector θ, the hypothesis function typically takes the form:

$$h_{(\theta)} = X\theta^T$$

An additional value of 1 is typically prepended to X, to account for vertical offsets.

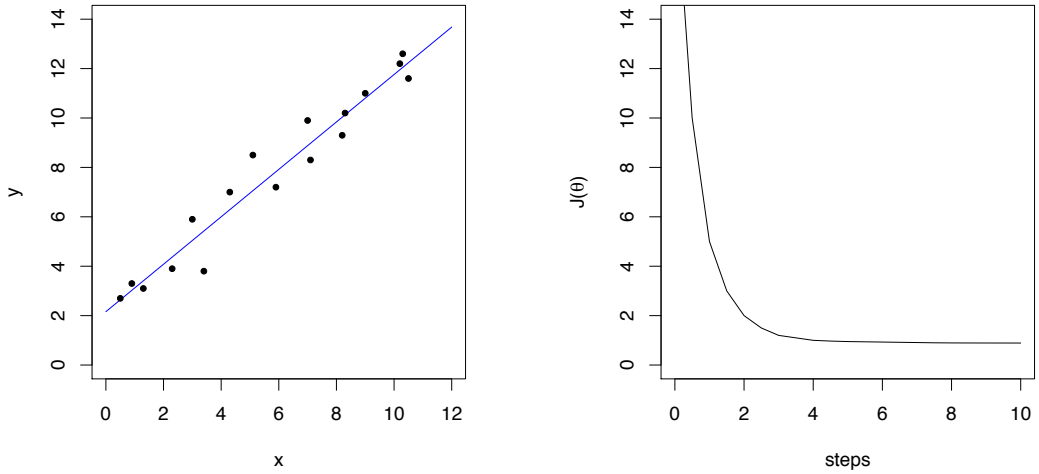
Calculating the values of θ can be done using matrix inversion, but this can be very computationally intensive, and in some cases impossible (for singular matrices). To avoid this, the θ values can be approximated using gradient descent [27]. This is done by first defining a mean squared error function $J_{(\theta)}$, which indicates how well the hypothesis matches the training data.

$$J_{(\theta)} = \frac{1}{2m} \sum \left(h_{(\theta)} - y\right)^2)$$

Then the algorithm will attempt to minimize $J_{(\theta)}$ by adjusting the values of θ based on the gradient of the cost function.

Figure 4.1 shows a prototyped implementation of gradient descent written in Haskell applied to a small dataset. Figure 4.1a shows the training data points and resulting hypothesis. It should be noted that the training data has not been normalised before running gradient descent, which explains the large number of steps required to obtain reasonably well-fitting θ values. Figure 4.1b shows how the cost function decreases quickly in the first few steps of the algorithm, and after this only decreases very slowly, eventually coming to a halt.

Applying linear regression to the procurement case described earlier is problematic. One might attempt to sort a set of suppliers by calculating a score for each supplier using a linear function. To do this, we would have to find the values of θ using gradient descent. Here we encounter a problem: the training data does not provide supplier scores, it only provides a partially sorted set (picked, not-picked). So the gradient descent algorithm does not have enough data to train on, making it unsuitable.



(a) *Hypothesis after 10.000 gradient descent steps.* (b) *Cost function for 10 gradient descent steps.*

Figure 4.1: *Linear regression and gradient descent applied to a simple dataset.*

4.3.2 Neural Networks

A neural network is used to solve classification problems based on a set of training data. It is represented as a graph of forward connected nodes which can be both acyclic and cyclic. Each node in the graph has a number of inputs with associated weights and typically an additional bias input and weight. The graph is divided into layers, where the outputs of a node in layer j connect to an input of all nodes in the next layer, i.

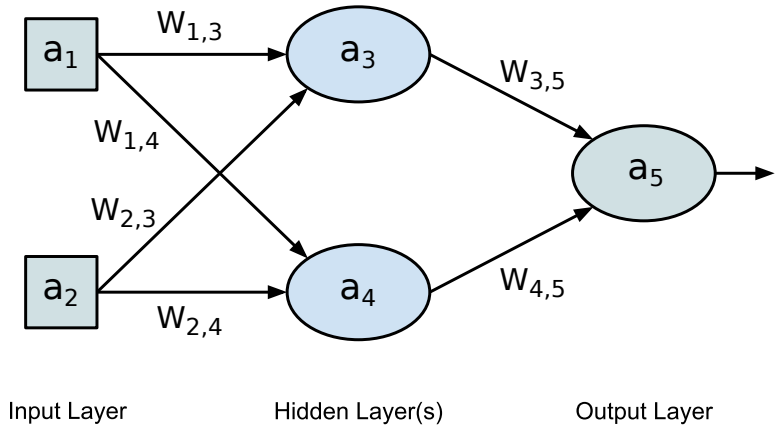


Figure 4.2: *Schematic view of a neural network with 1 hidden layer of 2 nodes.*

Each node on the input layer represents a single feature from the input data. The number of output nodes is determined by the number of possible results. For example, assuming fig. 4.2 is a well-trained net, there are two possible outcomes: either the upper or the lower output node is activated. The amount of the hidden layers and amount of nodes in these layers, are entirely up to the designer of the network [21].

The output of a node, a_i is computed using an activation function g. This is typically a sign/threshold or sigmoid function, resulting in a boolean (0 or 1) output value.

$$a_i = g\left(\sum_{j=0}^{n} W_{j,i} a_j\right)$$

Initially, the weights of the network are randomly assigned. The network will then have to be 'trained', using supervised learning with a set of input values and expected outcomes. This is done using backpropagation, a method in which the delta of each node's output is offset against the expected output for each entry in the training dataset. Based on this, the weight is adjusted and the process is repeated for the next entry.

The case that we're trying to solve does not entirely match the concept of Neural Networks, since it's a ranking problem rather than a classification problem. So without any modifications, the Neural Network approach is not suitable. However, some research has been done on using NN's as a sorting algorithm [4, 5]. In this modified use of NN's, the network is used in a sorting function, comparing the results of a pair of input values. This requires the backpropagation formula to be adjusted. Whether this could be successful in our case is unclear at this point. Additionally, it remains to be seen if it would produce a generic enough solution to the decision making problem.

4.3.3 Genetic Programming

Genetic Programming is a paradigm based on concepts taken from biology, such as evolution and natural selection. It is a specific version of a stochastic beam search and therefore works best when a solution can be modelled as a permutation of a finite number of components. Because it largely depends on randomness and combining different potential solutions, it is particularly effective in escaping local minima. This also means that the result of the program can be an unexpected, novel solution which still satisfies the technical criteria described in the fitness function. [14] Commonly, a genetic algorithm consists of the following steps: [22, 28]

1. Create an initial population of individuals by generating a random finite sequence over a finite alphabet. The sequence should describe all the properties of the individual and can be composed of different types of data, based on the domain. In a genetic algorithm it is typically a string or a bitarray. In a more high-level genetic program the alphabet is generally composed of logic statements.

2. Rate each individual using a fitness function.

3. Choose two random pairs (A, B) to create offspring, using the fitness determined at step 2. I.e. an individual with a higher fitness is more likely to be chosen than one with a low fitness.

4. Randomly choose a crossover point, c.

5. Create two new individuals:$A[0..c] + B[c..]$, and$B[0..c] + A[c..]$

6. Optionally mutate the newly created individuals by replacing one or more elements by a randomly chosen one. This should happen with a very low probability.

7. Repeat steps 2-6 until a certain fitness threshold or maximum iteration count has been reached.

Because of the nature of the case addressed in this thesis, the appliance of genetic programming seems to bring little benefits. Selecting the best supplier at any requisition situation is taken from a limited number of given options. Therefore, creating permutations of these options has no real-world value. The situation simply does not match the areas where genetic programming is beneficial. Furthermore, genetic programming does not address the entire problem, since it requires a fitness function to evaluate individuals. This would require knowing the utility of the features, which is not known beforehand.

4.3.4 Case-Based Reasoning

The main concept of Case-Based Reasoning (CBR) is to represent every problem as a case that contains description of the problem together with the most accurate solution [32]. As presented in fig. 4.3, the CBR process consists of four main stages [13]. When a new problem occurs, CBR tries to pattern-match it onto a set of historical cases in order to find the most similar problem that has happened in the past (retrieval stage). After this, the closest match is returned and its solution is applied onto the current problem (reuse stage). To assure the correctness of the case base, CBR requires human interaction in order to revise newly added cases (revise stage). Finally the new solution can be incorporated into the case base (retain stage) and enlarge CBR problem solving capabilities.

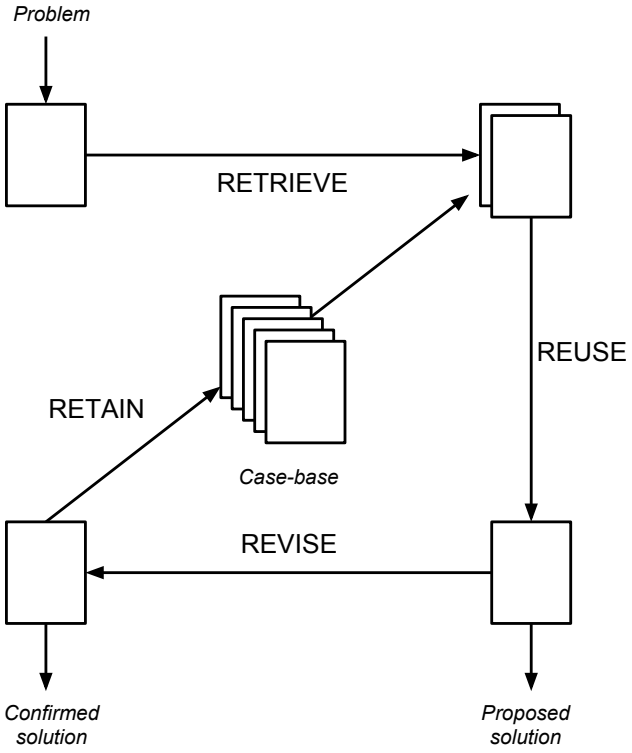


Figure 4.3: *The CBR Cycle*

In order for CBR to be effective, every case needs to be represented as a combination of a problem description reflecting a state of the world, a revised problem solution and outcome (the state of the world after the solution was executed).

CBR systems are especially useful in medicine (Medical Diagnosis Systems) [15, 8], Bioinformatics (Genetic classification system), Business (Call center automation).

Due to the nature of the problem, the solution will have to adapt the user's preferences rather than blindly pattern match the current state onto the historical case base. The fact that a particular supplier has been chosen in the past does not ensure that it will be the optimal choice for the current state. CBR does not compare the different available choices, but rather compares situations. For example, one might have a situation where a certain supplier is chosen a number of times because it is the best choice at that moment. If later on another supplier becomes available which is better than the previously picked supplier, CBR will not consider it as a better candidate when pattern matching.

4.3.5 Intelligent Agents

Intelligent Agents is one of the most widely used Artificial Intelligence approach. An agent can be thought of as anything that has the capability of perceiving the environment (through various

sensors) and the ability to act upon those observations [2]. It is a high level rational unit that by applying the Desire-Believe-Intention model, tries to mimic human reasoning process. The agent's reasoning process is driven by goals that try to be realised in the most efficient way, depending on its knowledge of the environment (Beliefs). By evaluating its current state and the state of the world, the agent simulates all the actions (Plans) and picks one that will move it closer towards its goal (Intention).

The concept itself focuses on structuring reasoning process rather than adding learning skills [23]. The only learning capability that can be obtained without altering the concept is done by editing the agent's plans. In order to add more complex learning methods, the agent would need to be provided with either a Neural Network or Case-Based Reasoning [9]. The key concept that make Intelligent Agents attractive is the ability to act autonomously, which allows them to work in distributed environments and form groups of agents.

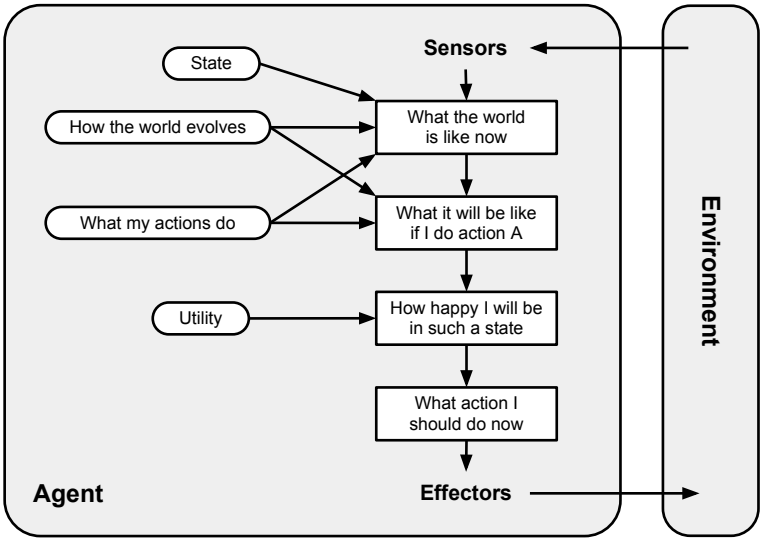


Figure 4.4: *Utility-Based Intelligent Agent design*

The application of Intelligent Agents is usually desired in situations when a system needs to mimic human behaviour. They are therefore widely used in areas such as Military, Robotics, Avionics and the Gaming Industry. Intelligent Agents are beneficial in situations when the rational process and emulating human thinking is important. As such, this approach addresses issues that are on a much higher level of abstraction than our problem. While picking the optimal supplier, the algorithm does not need to behave as human, but rather compute large volumes of historical data. It is clear that the algorithm would need to adjust continuously, and thus some form of learning is required. This is a property which agents do not provide.

4.3.6 Artificial Intelligence approaches comparison result

Even though the inspected solutions are well developed and widely used, they do not entirely fit our purpose. Some approaches, such as Intelligent Agents and CBR, aim at solving a slightly different problem. For others, like Neural Networks and Linear Regression, the major problem is related to the training data: the data available in our case is not sufficient for these approaches to work. The data obtained is only partially sorted into those suppliers that are picked and those that aren't. Since the order of the non-picked suppliers is not known, determining what is important among them is not possible. Due to the fact that all of these algorithms require data that is unknown at the time of training, there is a need to create a custom solution.

5 Solution

Our solution is an algorithm that calculates the utility values for each supplier, which can then be used to rank them. The supplier with the highest utility will be suggested to the procurer. Calculating a supplier's utility u is done in a similar fashion to using linear regression: a linear combination of a set of features s, and a set of weights θ:

$$u = s\ \theta^T = s_1\theta_1 + s_2\theta_2 + \ldots + s_n\theta_n$$

The weight values represent the relative contribution of each feature to the overall utility. For example, a procurer may be mainly interested in getting the lowest price, while fast delivery is of secondary importance. In this case, the weight on price should be higher, while the weight on delivery time should be lower.

5.1 Feature selection

Selecting the relevant features is one of the biggest challenges in suggesting the right supplier. Based on these selections the algorithm will calculate the purchaser's preferences. Therefore, the features need to represent all the most relevant factors that influence the decision making process. The omission of an important quality may result in miscalculated preferences and therefore incorrect suggestions. On the other hand, including many irrelevant features may have the same effect. Therefore our solution relies on the support of a domain expert determining the complete set of features.

It should be noted that the utility function described earlier requires features to have the same (positive) polarity. It should hold that for all continuous feature values a higher value means a higher utility. A straightforward way of achieving this is by letting a domain expert identify the features that should be inverted. For example, in the requisition case the price of an item should get a higher utility the lower its value is. In a later subsection (chapter 5.3.4), we propose a way of automatically determining the polarity of features.

5.2 Feature scaling

In order to make a fair comparison between features, they should all be normalised to a range of $[0, 1]$. The lowest value in each column becomes 0, the highest value becomes 1, and all values in between are scaled proportionally. This means that the relative distances between the feature values are preserved. This is represented by the following formula, in which c represents a column vector:

$$scale(c) = \frac{c - min(c)}{max(c) - min(c)}$$

For example:

$$S = \begin{bmatrix} 435 & 4 & 5 \\ 322 & 30 & 2 \\ 432 & 20 & 3 \end{bmatrix}$$

$$scale(S) = \begin{bmatrix} 1.0 & 0.0 & 1.0 \\ 0.0 & 1.0 & 0.0 \\ 0.973 & 0.615 & 0.333 \end{bmatrix}$$

5.3 Algorithm training

To calculate the utility of a supplier in a new requisition situation, the algorithm needs to know the feature weights. These weights represent what distinguishes the picked supplier from the

non-picked suppliers. Learning these weights from historical data is the goal of the training phase. This data contains item features for each supplier, supplier name, item number, the picked and non-picked suppliers.

The training phase iterates over a set of historical data, refining the weight values in each iteration. The more training examples are available, the more accurate the resulting weights become. However, it also means that the training data should be 'constructed' carefully. By this we mean that the decisions should be made considerately. After all, the learned weights will only be as good as the training data. We propose several approaches to calculate the feature weights.

5.3.1 Notation

Throughout this chapter we'll use the following notation:

- Given a matrix A:
  - $A_{x,\_}$ denotes the x^{th} row of A.
  - $A_{\_,y}$ denotes the y^{th} column of A.
  - $A_{x,y}$ denotes the element in A at position x, y.
- Let $s \in \mathbb{R}^n$ be a row-vector with feature values (e.g. price, delivery time, etc.) for a specific part at a single supplier at a single point in time. n denotes the number of features in s. E.g.
$$s = \begin{bmatrix} price & delivery\ time & \dots & s_n \end{bmatrix}$$
- Let S be a m x n matrix, representing supplier's feature values for a single procurement situation, where m is the number of suppliers available for an order. Each row in the matrix is an s-vector. E.g.
$$S = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{bmatrix} = \begin{bmatrix} s_{1,price} & s_{1,delivery\ time} & \cdots & s_{1,n} \\ s_{2,price} & s_{2,delivery\ time} & \cdots & s_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m,price} & s_{m,delivery\ time} & \cdots & s_{m,n} \end{bmatrix}$$
- Let $s_{picked} \in S$ denote the supplier that was chosen to be ordered from.
- Let the training data T be a set of tuples, each being a procurement situation S combined with an integer value *picked* indicating which supplier in S was chosen. o is the total number of orders. E.g.
$$T = \{(S_1, picked_1), (S_2, picked_2) \dots (S_o, picked_o)\}$$

5.3.2 Algorithm A - Sum of picked values

The first approach relies on comparing all suppliers on the same scale and looking at the position of the supplier that was picked.

As described earlier, the feature values are normalised to a scale of $[0, 1]$, where a higher value means that the feature gives a higher utility. Doing so gives a comparison between the picked supplier's feature values and the extremes in the set of suppliers to pick from. If a feature's values of the picked supplier are consistently high across a number of training examples, it means that this feature must be of importance to the decision. Similarly, if the feature's values are consistently low or varying, they must not be significant for the decision. So by summing the normalised feature values of the picked supplier in a training set, we obtain the feature weights:

$$\theta = \sum_{j=1}^{o} (T_{j,picked})$$

5.3.3 Algorithm B - Difference from average of picked values

To refine the approach described in the previous section, we introduce the concept of calculating the weights by taking the average feature value into account. The feature values are first normalised to a range of $< 0, 1 >$. In contrast to the previous approach, this approach looks at the distribution of all candidates in order to find features that stand out from the rest rather than just looking at the extreme values.

Consider the training example in figure 5.1. The value of feature p for supplier $sup5$ is very low, while most other alternatives have a value just below the picked supplier $sup3$. Using the first approach, this would result in a high weight on this feature, even though it doesn't 'stand out' among the alternatives. By calculating distance of the picked value to the average (the blue line), the algorithm would produce a lower weight, presented as x.

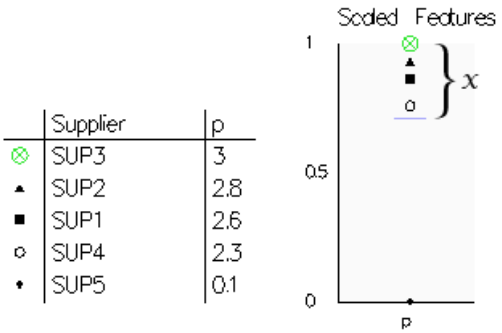


Figure 5.1: *Training example with non-equidistant feature distribution.*

The following steps are taken to calculate the feature weights:

1. Calculate the average value of each column in S. ($\overline{s}$ is a row-vector)

$$\overline{s} = \frac{1}{n} \sum_{i=1}^{n} S_{i,-}$$

2. Subtract $\overline{s}$ from s_{picked}.

$$\delta = s_{picked} - \overline{s}$$

3. Apply steps 1, 2 for all orders in T, resulting in a matrix Δ with the differences for each feature at each order.

$$\Delta = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \dots \\ \delta_o \end{bmatrix}$$

4. Calculate θ by taking the sum of each column in Δ, resulting in a row-vector.

$$\theta = \sum_{j=1}^{o} \Delta_{j,-}$$

All steps combined gives:

$$\theta = \sum_{j=1}^{o} \left(T_{j,picked} - \frac{1}{n} \sum_{i=1}^{n} T_{j,i} \right)$$

5.3.4 Algorithm C - Automatic polarity calculation

In the previous two approaches, the polarity of each feature would have to be known in advance. To avoid this, we create two new features to replace each individual feature. These represent the features' values with different polarities: a positive polarity means that a higher value gives a higher utility and a negative polarity means that a lower value gives a higher utility.

$$p_+ = p \tag{5.1}$$

$$p_- = 1 - p \tag{5.2}$$

$$\theta = \sum_{n}^{o} T_{n,picked} \tag{5.3}$$

$$u_p = \begin{cases} \theta_{p_+} >= \theta_{p_-} & = p_+(2\theta_{p_+} - 1) \\ \theta_{p_+} < \theta_{p_-} & = p_-(1 - 2\theta_{p_+}) \end{cases} \tag{5.4}$$

The positive polarity is created by taking the features' values without modification (1). The negative polarity is the distance to the highest value (2). The weight of each feature is calculated in the same manner as in Approach 1: by taking the sum of the feature values for the picked supplier (3). This results in a double amount of weights, corresponding to the positive and negative polarities. The most desired quality among p_+ and p_- will also have the highest accumulated sum.

When calculating supplier scores (4), only one of the polarized features is used, namely the one which has the highest feature weight. The feature value is multiplied by a weight obtained by taking the absolute of the difference between the polarized weights. This is done to limit the influence of the weights if the polarization is not fully certain. The final utility is a sum of the utilities for all features, as explained in a previous section.

6 Prototype overview

The results of the proposed algorithms have been calculated and presented with the use of an interactive Haskell program. The program provides a fine-grained overview of the algorithm's intermediate calculation, to ease the analysis and verification of the results. The user interface is divided into three main sections, as shown in figs. 6.1 to 6.3.

The scatter plot in fig. 6.1 shows the scaled feature values, making the interpretation of their relative positions easier. The table in fig. 6.1 presents the raw feature values (p, q, etc.) of the picked supplier together with its alternatives. Each supplier is assigned a unique symbol that helps to trace them throughout different training examples. To be able to distinguish the picked supplier from the others, its symbol is shown in green. The scores for the suppliers are shown on the right-hand side of the table in form of a bar chart together with corresponding numerical values.

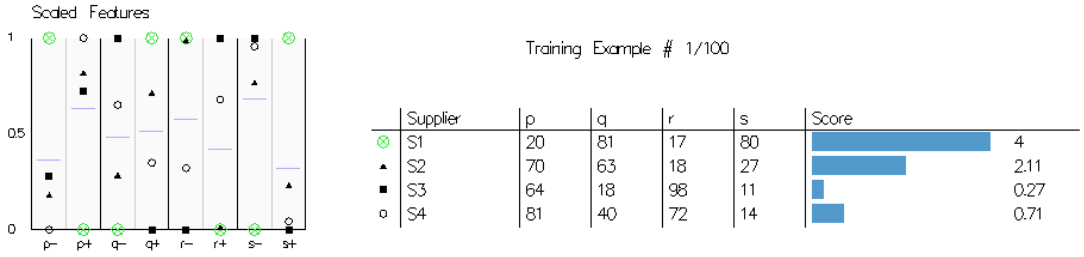


Figure 6.1: *Input and results charts.*

Figure 6.2 represents different calculation steps. The leftmost bar chart presents the feature weight of a current training set. This reflects the relative feature importance calculated based on

the current training example. The second chart from the left presents the cumulative of feature weight up to the current training example. This shows how each training example influences the final weights. The middle-right chart shows the feature weight calculated after finishing the training phase. The rightmost diagram displays the error rate plotted against the number of training examples used to train the algorithm. This metric is described in more detail in section 6.2.

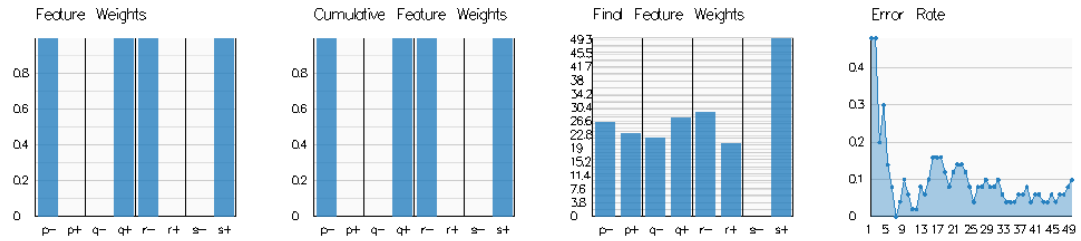


Figure 6.2: *Weights calculation and error rate section.*

The last section, fig. 6.3, shows the scores for all supplier throughout the entire training-set. This can be used for trend analysis, as discussed in Future Work (section 11.2).

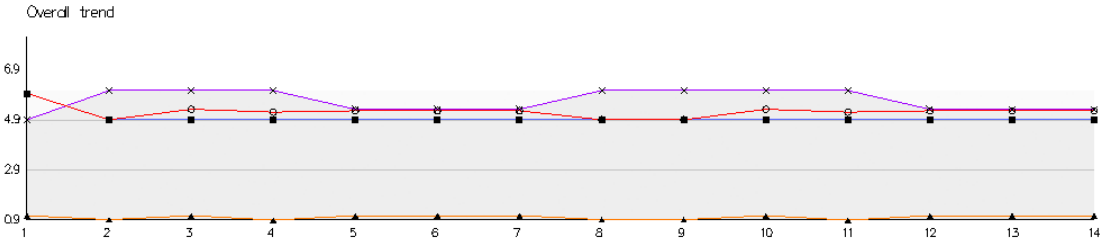


Figure 6.3: *Trends overview. The vertical axis represents the supplier score, the horizontal axis represents the training sets.*

For the documentation of the prototyped Haskell implementation, see appendix E.

6.1 Input data

The data fed into the algorithm is stored as comma separated values and structured as presented in table 6.1. The first column represents the order id used to differentiate the training examples. The other columns represent the suppliers' specific information for example name, lead time and price. The program will consider the first row in a training example to be the picked supplier.

O1	S1	3	1.15
O1	S2	15	1.18
O1	S3	3	1.37
O1	S4	3	1.37

Table 6.1: Example training data

6.2 Error metrics

An essential quality of the algorithm is its error-rate: the ratio between incorrect and correct results. During the training phase the error-rate indicates how well-trained the algorithm is. As the error-rate nears zero, the algorithm has learned enough to recreate most training example decisions, and no further training would be required.

For each training example we compare the picked supplier of the training data and the algorithm's result. The training set provided to the algorithm is divided into two subsets: the first is used to learn the weights, and the second is used as a control group. The error metrics are calculated from the latter subset.

7 Results

Various tests were carried out using the prototyped versions of the algorithm to compare their qualities. In this chapter we will present the results in form of final weights and error rates of algorithms calculations using different data sets.

The algorithm variations were tested using real world data sets acquired from two sources: a large manufacturing company using IFS Applications and a hardware price comparison portal. The decisions made in the former set were created by purchasers from the company, while the latter set did not contain decision information. This had to be introduced manually.

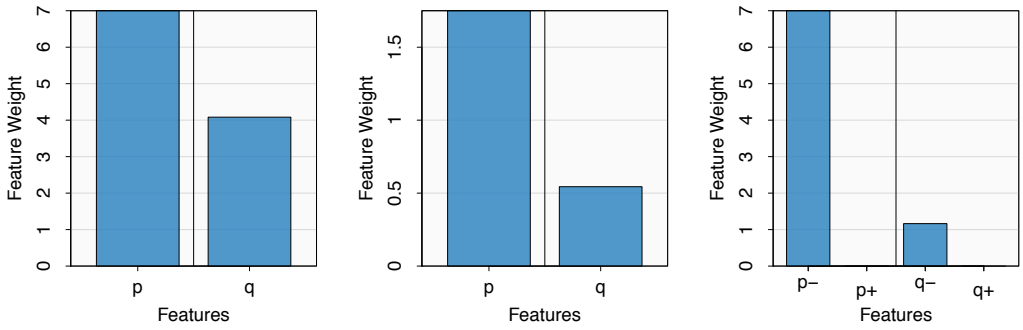
Furthermore the algorithm were tested using artificially generated data to verify whether algorithms keep their properties in presence of fluctuating. This also allowed tests on training sets with a large number of features, alternatives, and training examples. Additionally, the most relevant features for the selection of the best alternatives in the training examples could be controlled.

Finally, sensitivity analysis was performed to test whether the algorithms could recover from errors in the training data.

7.1 Tests on data from manufacturing company

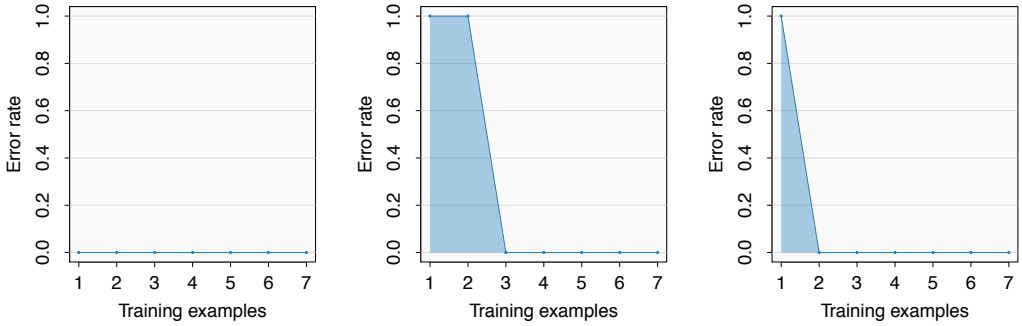
The manufacturing company supplied an extract of their procurement database to be used as training data for our algorithms. It contained a wide range of procurement data, many of which may be selected by a domain expert as features influencing a procurement decision. However, for the purpose of this thesis, the training-set was limited to price (q) and leadtime (p). Both features were set to have a reverse polarity, such that a lower price gives a higher utility. The data set consisted of 14 orders with 4 supplier alternatives.

Figures 7.1a to 7.1c show that the weights found by all three algorithm variations exhibit similar characteristics. All show that a low leadtime is the most important feature, while price is of secondary importance, with a significantly lower weight. All three algorithms perform well, achieving an error-rate of 0 either immediately or after a couple of training-examples (figs. 7.2a to 7.2c).



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure 7.1: *Feature weights for different algorithms, using manufacturing training set.*



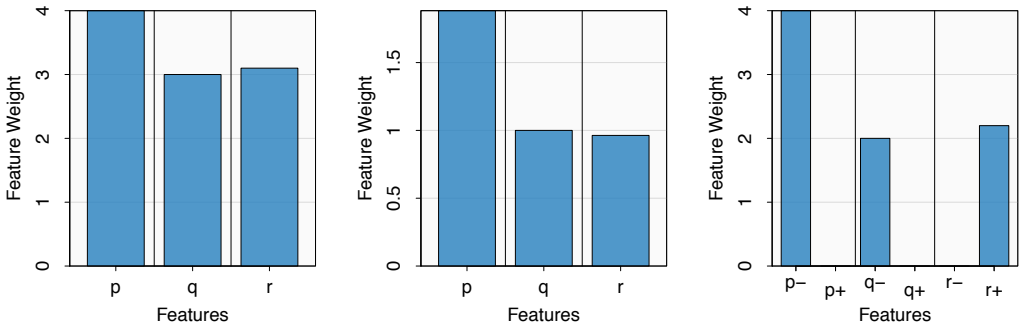
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure 7.2: *Error-rates for different algorithms, using manufacturing training set.*

7.2 Tests on data from hardware price comparison portal

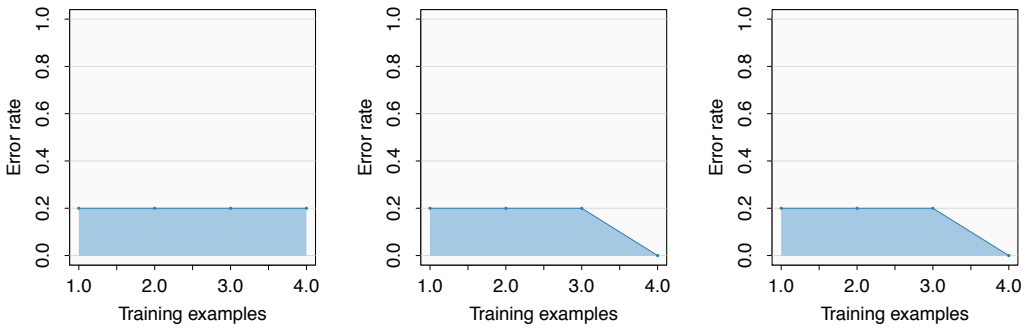
The data obtained from hardware comparison portal consisted of item price, delivery time and rating. The suppliers represent different hardware retailers. The inspected computer hardware item was an internal memory module. The features price (p) and delivery time (q) were set to have a reversed polarity. The training data consist of 9 orders with 3 supplier alternatives.

Figure 7.3 presents that algorithm A does not provide a clear distinction in importance between the features. On the other hand, algorithms B and C determine that the price is by far the most important feature, whereas the lead-time and rating are less important. The weights calculated by algorithm C clearly show that determining the polarity of features works: price and lead-time have a negative polarity, while the rating has a positive polarity.



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure 7.3: *Feature weights for different algorithms, using hardware training set.*



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure 7.4: *Error-rates for different algorithms, using hardware training set.*

7.3 Tests on artificial data

To further analyse the error-rates and feature weights of the algorithm variations, 5 artificial training-sets were created. Every training-set consisted of 100 training examples o, with 4 alternative suppliers m. With each generated training-set, the number of features was incremented by 1. The feature values were randomly and uniformly selected from a range $[0, 100]$. One feature column per training-set was randomly selected to be the quality on which the decision was made. For each training example, the supplier with the highest feature value was selected as a best choice.

As in the case of real data, the training-sets were split into two halves. The algorithms were trained using the first 50 training examples. The second half of the training-set was then used to verify the correctness of the learned weights. The decisions made by the algorithm were compared with those made when generating the training-set. The error rates in table 7.1 show that all algorithms perform without a single error when there is only one feature to base the decision on. As the number of features increases, the error rate rises. In particular for algorithm A exhibits a significantly higher error rate than B and C. This phenomenon is caused by the random numbers used to generate the data. Because of the fact that algorithm A is using sum of the weights it receives significantly high utility for unimportant qualities. As a result of this, suppliers with a low value for important qualities, but high values for unimportant qualities may receive a higher score than the picked supplier's score.

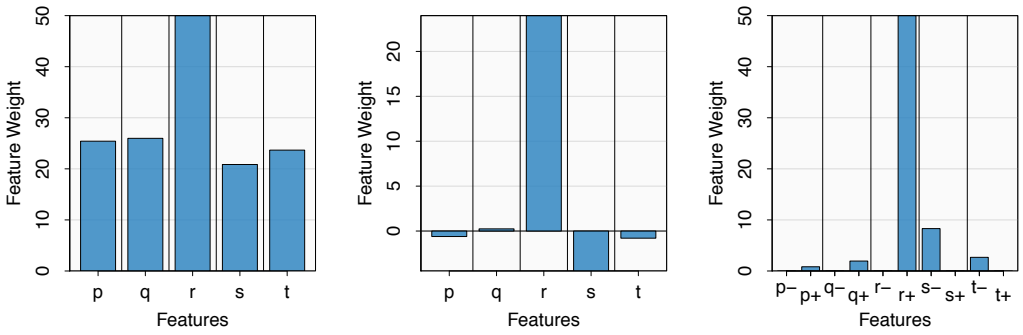
Features (n)	1	2	3	4	5
Algorithm A	0.00	0.20	0.36	0.36	0.50
Algorithm B	0.00	0.02	0.14	0.20	0.04
Algorithm C	0.00	0.02	0.20	0.18	0.04

Table 7.1: Error rates for decisions made on the control-set, with $o = 50, m = 4$.

Algorithm B ensures that features occurring frequently below the average (i.e. those that are not of importance to the purchaser) do not receive high value. Every feature value that is below the average is subtracting its weight value from the cumulative weights. This prevents high weights for unimportant features. On the other hand, algorithm C while trying to determine feature polarities, takes the absolute difference of negative and positive polarities. Because the generated feature values are uniformly distributed, the polarities of unimportant qualities counterbalance each other.

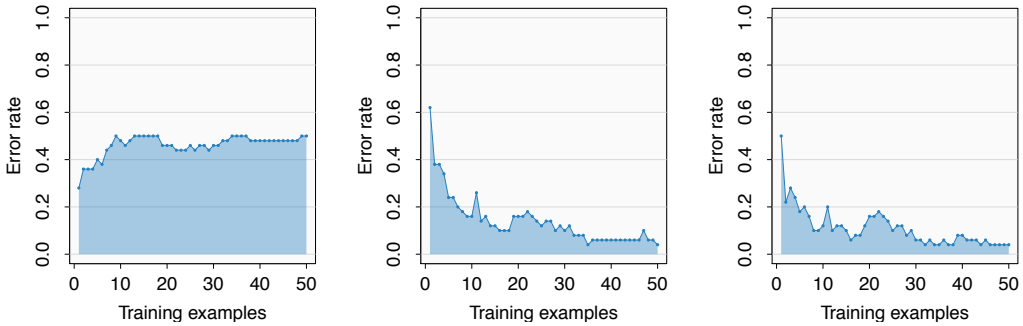
To gain a better understanding of the properties of the algorithms while training, the error rates were plotted against the number of training examples as shown in figs. 7.6a to 7.6c. For the reasons described above, algorithm A keeps performing poorly, even when a large number of training examples is used. Algorithms B and C show a good learning progress, eventually reaching an acceptable error rate. The similarity between the plots of these algorithms is a result of the similarity in computing weights for unimportant features. Algorithm B progresses slightly more gradual than algorithm C. The latter is more susceptible to fluctuations in individual training examples.

The same tests were performed on a training set where the combination of two features determines the best supplier. The results of these tests were similar to the tests with 1 important feature. The graphs of these tests can be found in appendix B.



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure 7.5: *Feature weights for different algorithms, using artificial training set.*



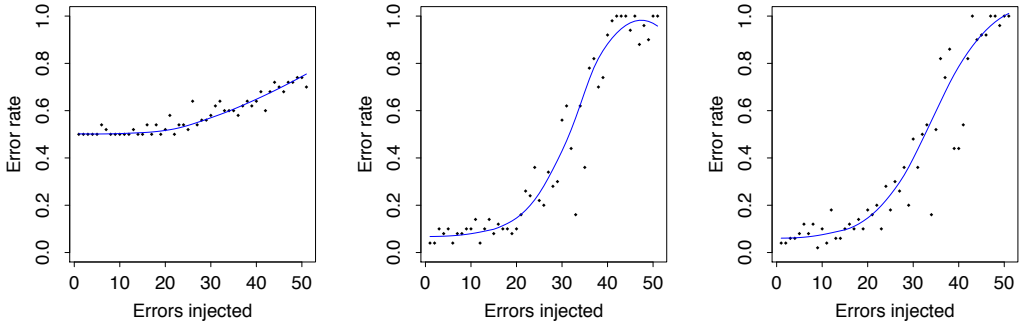
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure 7.6: *Error-rates for different algorithms, using artificial training set.*

7.4 Sensitivity analysis

In real-world situations it is unavoidable to have errors in the training data. These errors should not have an effect on the outcome of the algorithms. To analyse the robustness of the algorithms, the error-rates have been calculated in the presence of artificially generated errors. For each training set, decision errors were emulated on randomly selected orders in the set. The picked supplier for these orders was changed to one of the non-selected suppliers. Figure 7.7 shows the error rates for training sets where up to 50 errors were inserted.

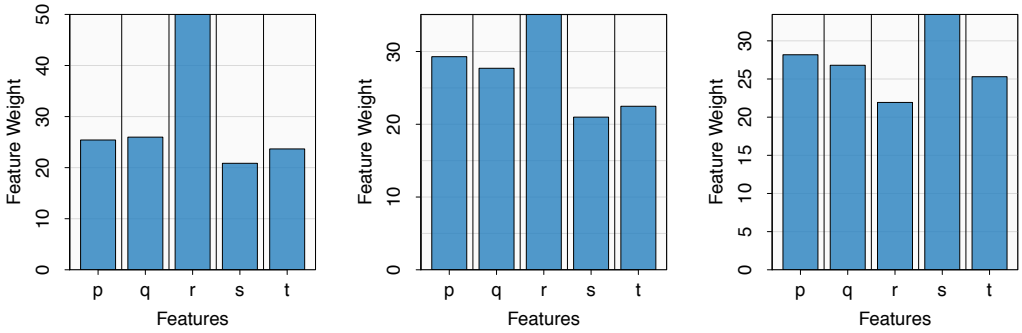
Figure 7.7a shows that algorithm A continues performing poorly after error insertion. The error rate is linearly increasing with every data mistake introduced. Therefore the solution A does not provide any fault tolerance. On the other hand, the sensitivity diagrams of algorithms B and C (figs. 7.7b and 7.7b) resemble a sigmoid shape. This means that their desired properties are preserved, even in the presence of errors.



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

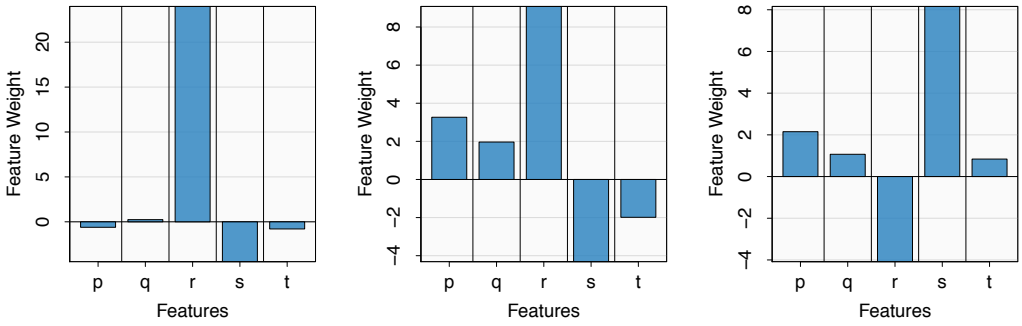
Figure 7.7: *Algorithm sensitivity using a training-set with 5 features.*

When half of the decisions in the training-set are incorrect, the weights calculated by the algorithms still preserve their characteristics, as seen in figs. 7.8b to 7.10b. With every additional error introduced to the training-set, the feature weights become further removed from the optimal.



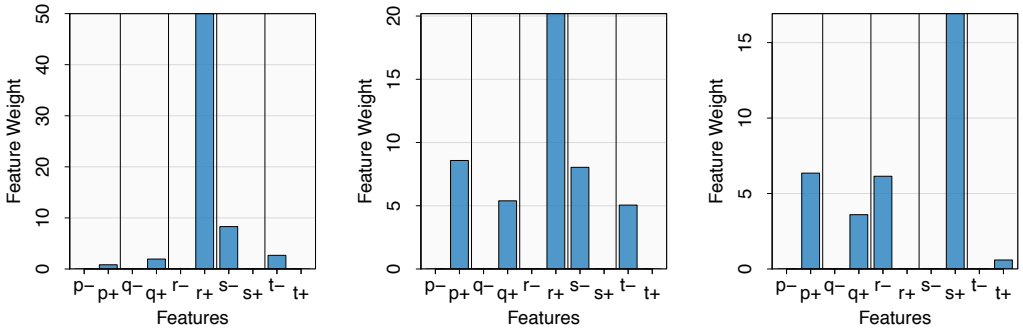
(a) *0 errors* (b) *25 errors* (c) *50 errors*

Figure 7.8: *Weights for algorithm A, with varying number of injected errors.*



(a) *0 errors* (b) *25 errors* (c) *50 errors*

Figure 7.9: *Weights for algorithm B, with varying number of injected errors.*



(a) *0 errors* (b) *25 errors* (c) *50 errors*

Figure 7.10: *Weights for algorithm C, with varying number of injected errors.*

8 Discussion

In the previous chapter the algorithm variances were tested using both real-world and artificially generated training-sets. Even though algorithm A performs well in the real-world data tests, it shows significant deficiencies in the artificial data tests. With error-rates of up to 0.5, it is not going to be reliable as an autonomous decision making system.

On the other hand, algorithm B and C performed well in all tests, in some cases with error-rates close to 0. The performance of both algorithms is very similar, and they could therefore be considered as equivalents. However, it should be noted that algorithm B requires the polarity of the input data to be known and corrected before running the algorithm, as described in section 5.1. Algorithm C does not have this requirement, since it determines the feature polarities during the training phase. Considering this, using algorithm C over algorithm B is preferable.

8.1 Input data requirements

In order to work correctly, the algorithm needs to meet several prerequisites. Firstly, all the factors influencing the decision need to be captured and represented as a numeric values. This step is the most crucial for the algorithm to work correctly. Preferably the features that are not of importance should be omitted in the training set. However with significantly large training set the algorithm shows resilience to this problem, which was tested with artificially generated data.

Secondly, the data used in training phase need to be up-to-date and the decisions made need to be consistent. Therefore it is advised to train the algorithm on current cases and switch of the learning phase when the decisions start to produce satisfying results. Even though the algorithm keeps performing well in the occurrence of errors, the decisions should exhibit consistent importance properties and the exceptional cases should be omitted.

Thirdly, the unimportant features should exhibits significant oscillation in their values. This is a case because the algorithm treats consistently high values as significant to the decision making process. Therefore if those values are high in a large amount of training cases then they might be mistakenly considered as important.

8.2 Study limitations

For various reasons, the tests performed on real-world data in this study had a limited scope. It turned out that obtaining a suitable data-set for the tests was harder than initially estimated. Even though there was a large quantity of data from various sources, very few matched the requirements described above. In particular the correctness of the decisions reflected in the data proved problematic. Most decisions appeared to be made using the predefined 'main supplier' functionality. We believe that this is in fact an indication of the need of a more sophisticated solution, such as the algorithms developed for this study.

The data freshness (i.e. the degree to which the available data reflects the current real-world situation) also proved to be an issue. For example, the data-set from the manufacturing company contained many purchase lines where it seemed that only values of the chosen supplier were updated. However, because the data-set is constructed from historical, it is very difficult to determine whether this was actually the case, or if it was a coincidence.

Finally, the number of suitable training examples with a sufficient number of alternatives was limited. This made extracting a meaningful training-set from the real-world data difficult. To increase the size of the training-set, we bundled items with similar characteristics. This increased the number of examples to train on, resulting in better outcomes.

Because of the scope of this study, the number of features used for the supply chain management case was limited. The feature selection has been done by a Supply Chain Management Expert, who named the most relevant features for the procurement decision making process. In real-world use of the algorithm the set of features could be expanded by adding features such as: quality metrics and delivery time reliability (i.e. does the supplier deliver ordered items on time).

8.3 Algorithm limitation

The testing phase has revealed that none of the algorithm alternatives reach an error-rate of zero. This is caused by certain rare combinations of feature weights and values. An alternative with a combination of high values for unimportant features can accumulate a higher utility than an alternative with a high value for important features.

$$\theta = \begin{bmatrix} 1 & 0.25 & 0.25 \end{bmatrix}$$

$$S = \begin{bmatrix} 90 & 10 & 50 \\ 80 & 20 & 90 \\ 50 & 50 & 70 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.75 & 0.25 & 1 \\ 0 & 1 & 0.5 \end{bmatrix}$$

$$u = \begin{bmatrix} 1 \\ 1.0625 \\ 0.365 \end{bmatrix}$$

The tests with the artificial data have shown that this problem can be largely mitigated by prolonging the training period.

9 Generalization

The solution presented could be used to help automate various operational decisions which require a choice between several candidates represented as a finite set of properties. Furthermore the algorithm is highly scalable in terms of number of alternatives and features. Various business cases requiring a fairly large number of features can be computed without any adjustment to the algorithm.

9.1 Within IFS Applications

The procurement case researched in this thesis is one of the examples how automation can be introduced into IFS Applications, as part of the 'IFS Supply Chain' in fig. 9.1. The proposed algorithm generating suggestions based on users preferences can be used either fully automatically, or be used to provide a suggestion to the purchaser. Additionally, the business intelligence tools built around the algorithm can be used to aid evaluation of suppliers (see also section 11.2).

Implementing the algorithm as part of the suite should not require major changes to the existing architecture. However, it may require preserving additional training data, which is currently discarded. For example, the information related to suppliers that were not picked and quality measurements.

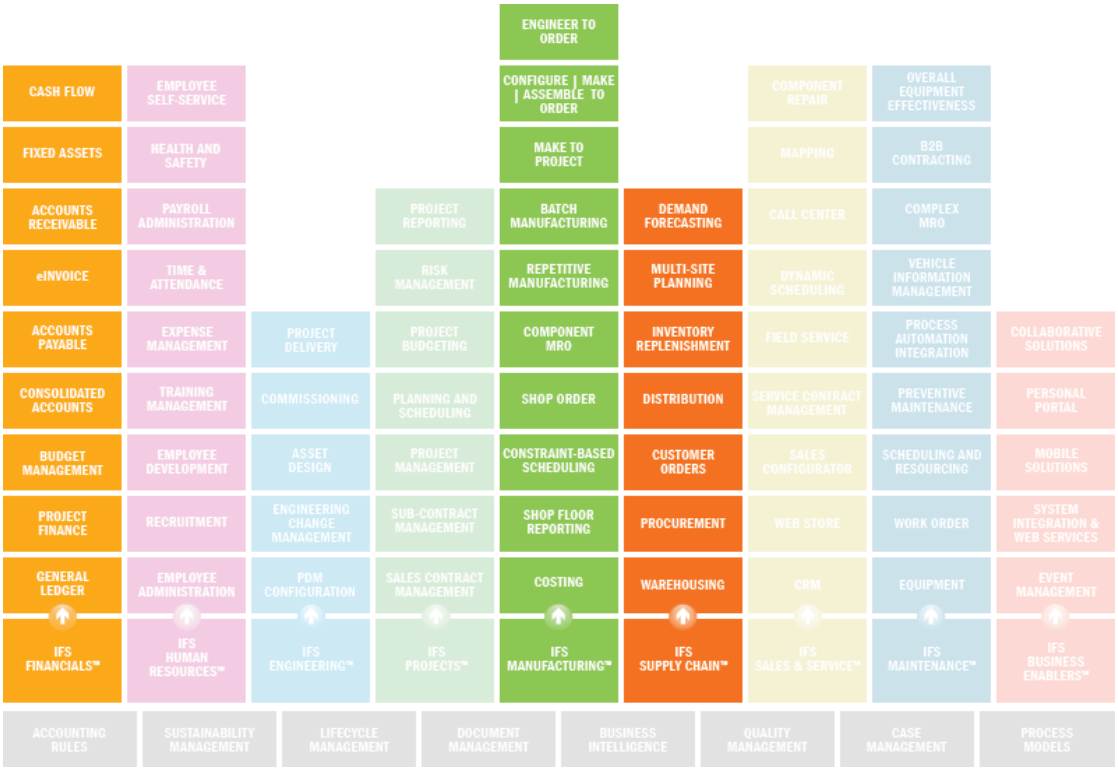


Figure 9.1: *Potencial areas within IFS Applications where the algorythm can be introduced [7].*

Another case where this algorithm can be applied is 'IFS Manufacturing' (fig. 9.1). For example, when allocating resources during production line planning, the algorithm could aid in selecting the best resource for a particular job. This can be done based on various features, such as availability, reliability, production speed, deadlines and other requirements. The algorithm will learn preferences of resource allocation and apply them to new cases. The advantage of using the algorithm over traditional planning would be that it could consider more features than a human planner.

In 'IFS Financials', the algorithm could help to determine how to delay the payment of incoming invoices. It is common practice for companies to try to pay invoices as late as possible, without damaging the relationship with the creditor. Similarly, the company itself will have outstanding bills to their clients. So the longer they can refrain from paying their bills, the more money they save. Determining how long to delay the payment can be automated using the following properties as a features and applying them to the algorithm: amount to be paid, the cost the delay, the due date and the frequency of the situation. In this case, the alternatives the algorithm will pick from represent different delay periods.

9.2 Other applications

Additionally, ERP systems are not the only place where the algorithm could be used. An example of the case outside the IFS Applications could be the insurance industry. The algorithm could aid to determining the action when processing insurance claims. The system could look into factors such as the the clients claim history, income, insurance rate, area of residence etc. The output would present the ranked actions that could be undertaken, which can in have a big impact on the speed of claim processing.

In addition to making decisions, the algorithm could also be useful as an evaluation tool, running in parallel with the regular decision making process. In this case employees will continue making decisions manually, while the algorithm is used a double-check. By doing so it can be used to compare the human decisions with the software suggestions, and provide performance

rankings. This appliance of the algorithm is not restricted to a particular domain.

10 Conclusion

In this thesis, we investigated how business decision making can be automated with the use of AI in the context of ERP systems. We've investigated which criteria influence the suitability of a business decision for automation, and how this automation can be achieved. Specifically, we've looked at how the relative importance of features influencing a decision can be determined and how this can be used to automate the decision making process.

Which criteria make an operational business decision a good candidate for automation using AI?
The theory review (chapters 3 and 4) has shown that in order to automate the decision making process each decision has to be represented by a set of features that influence it. These features have to be expressed as *continuous values* which can be used in mathematical calculations. Additionally, it was found that in order for these calculations to be correct with respect to the real-world situation, the *features need to represent only those factors that are relevant to the decision making process.* Therefore, selecting these features will require specific domain knowledge. In our project this was done by consulting a supply chain management expert, who provided the significant criteria when picking the optimal supplier.

To train the algorithm, *decision data has to be preserved.* This data should contain all features used to make the decision as well as the outcome of the decision. The tests on real-world data obtained from a manufacturing company have shown that the current version of IFS Application contains just enough data to fulfil this requirement.

How can the relative importance of features contributing to an operational decision be expressed?
Once it is known which criteria contribute to the decision making process, the relative importance of these factors has to be determined. To make a fair comparison among features, they are normalised to an equal scale. Based on the choice among alternatives, we calculate the feature utility and determine its polarity. The weights learned from all training examples are then accumulated to express the overall feature weights, as described in chapter 5.

With this relative importance, how can an operational business decision be made automatically?
The feature weights learned during the training phase can be used to calculate a score value for each alternative, determining their ranks. It follows that the alternative with the highest score should be selected as the decision outcome. An ERP system can be configured to use the algorithm's outcome to automate the decision making process.

The tests on various data-sets (sections 7.1 and 7.3) have shown that this is a solution which is capable of making decisions accurately. Sensitivity analysis (section 7.4) has shown that the algorithm preserves its qualities, even in the presence of errors in the training data.

Our prototyped solution shows that applying AI to automate operational business decisions is achievable, and can significantly reduce the time required for making these decisions. It is also plausible that the introduction of an AI algorithm will improve the decision quality in situations where manual decision making is too time consuming to carry out carefully.

11 Future work

Within this chapter we will present some possibilities of extending the current solution with features that were outside of the scope of this thesis.

11.1 Features with discrete values

During this project we've only considered features with continuous values. We can imagine that there are scenarios where not only continuous but also discrete valued features influence a decision. Handling these will require some adjustments to the scaling algorithm. A first prerequisite is that the positions of the discrete values in the feature matrix are known. E.g. columns 3 and 5 are discrete represent discrete valued features, the others are continuous values. Secondly, all possible feature values have to be known to be able to determine the value range. Finally, the discrete values have to be mapped to numeric, continuous values before using the algorithm.

Example: Assume the first feature in the training set represents a color with one of four different values: red, green, blue, yellow. The mapping between these colors and numeric values can be:

$$color = \begin{bmatrix} red = 0 \\ green = 1/3 \\ blue = 2/3 \\ yellow = 1 \end{bmatrix}$$

Similar to the polarity features, we create new features for each of the possible discrete values n. p_n represents the utility of

$$P_n = 1 - |v - color_n|$$

These are then used in the same way as regular features. However, separating them from the polarity calculation would require additional work.

11.2 Trend analysis

Currently, purchasers deal with ad-hoc, day-to-day purchase decision making. Because this takes more time than they actually have available, they will only focus on decisions that they think are the most important. The decisions deemed less important are done in a suboptimal, preconfigured way and are evaluated yearly. Using the algorithm's resulting scores for suppliers and applying trend analysis to this, the system could identify which suppliers or requisitions need more attention. For example, if the preconfigured supplier's scores are gradually decreasing it may be a indication that the agreement should be reviewed. By doing this, the purchasers can focus their effort on those parts which are most valuable to the company.

Taking this a step further, the trend analysis can be used to automate a tactical decisions, by inspecting the gradients of the trending diagram.

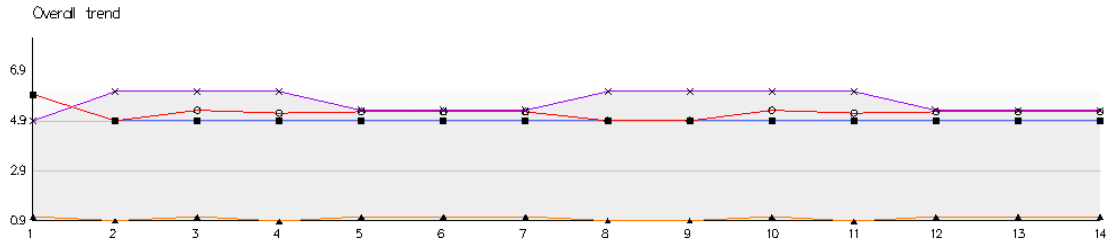


Figure 11.1: *Example of a trend graph.*

References

[1] J. R. Anderson. *The Architecture of Cognition*. Cambridge, MA: Harvard University Press, 1983.

[2] D. C. C. Annapurna Valluri. "Agent learning in supplier selection models". In: *Decision Support Systems* 39.2 (2005), pp. 219–240.

[3] D. Berkeley and P. Humphreys. "Structuring decision problems and the 'bias heuristic'". In: *Acta Psychologica* 50.3 (1982), pp. 201 –252.

[4] C. Burges et al. "Learning to rank using gradient descent". In: *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM, 2005, pp. 89–96.

[5] C. J. C. Burges. *From RankNet to LambdaRank to LambdaMART: An Overview*. Tech. rep. Microsoft Research, 2010.

[6] I. Gilboa and D. Schmeidler. "Case-based knowledge and induction." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part A* 30.2 (2000), pp. 85–95.

[7] *IFS Applications - Full Suite ERP*. `http://www.ifsworld.com/en-gb/solutions/product/`. Accessed: 2012-10-04.

[8] C. M. Isabelle Bichindaritz. "Case-based reasoning in the health sciences: What's next?" In: *Artificial Intelligence in Medicine* 36.2 (2006), pp. 127–135.

[9] M. A. P. Juan M. Corchado. "Development of CBR-BDI Agents". In: *IJCSA* 2.1 (2005), pp. 25–32.

[10] J Kacprzyk, ed. *Studies in Computational Intelligence* 97 (2008).

[11] R. L. Keeney. "Expert judgment and expert systems". In: ed. by J. L. Mumpower et al. Springer-Verlag, 1987. Chap. Value-driven expert systems for decision support, pp. 155–171.

[12] G. Klein. "A recognition-primed decision (RPD) model of rapid decision making". In: *Decision Making in Action: Models and Methods*. Norwood: Ablex Publishing Corporation, 1993, pp. 138–147.

[13] J. L. Kolodner. "An introduction to case-based reasoning". In: *Artificial Intelligence Review* 6.1 (1992), pp. 3–34.

[14] J. Koza, F. Bennett, and O. Stiffelman. "Genetic Programming as a Darwinian Invention Machine". In: *Genetic Programming*. Ed. by R. Poli et al. Vol. 1598. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1999, pp. 651–651.

[15] R. C. W. Monique Frize Lan Yang and A. M. O'Connor. "Conceptual framework of knowledge management for ethical decision-making support in neonatal intensive care". In: *IEEE Transactions on Information Technology in Biomedicine* 9.2 (2005), pp. 205–215.

[16] A. Newell. *Unified theories of cognition / Allen Newell*. Harvard University Press, Cambridge, Mass., 1990.

[17] J. Pohl. "Cognitive Elements of Human Decision Making". In: *Studies in Computational Intelligence* (2008).

[18] J. Pomerol and F. Adam. "Understanding Human Decision Making – A Fundamental Step Towards Effective Intelligent Decision Support". In: *Studies in Computational Intelligence* (2008).

[19] K. R.L. *Value-Focused Thinking. A Path to Creative Decision Making*. Cambridge: Harvard University Press, 1992.

[20] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003.

[21] S. J. Russell and P. Norvig. “Artificial intelligence: a modern approach”. In: Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003, p. 737.

[22] S. J. Russell and P. Norvig. “Artificial intelligence: a modern approach”. In: Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003, p. 133.

[23] S. J. Russell and P. Norvig. “Artificial intelligence: a modern approach”. In: Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003, pp. 31–51.

[24] L. Savage. *The Foundations of Statistics*. New York: Dover Publications, 1972.

[25] R. C. Schank. *Dynamic memory - a theory of reminding and learning in computers and people*. Cambridge University Press, 1983, pp. 1–234.

[26] H. Simon. “Administrative behaviour”. In: Free Press, New York, 1997, pp. 93–94.

[27] J. Snyman. “Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms”. In: Applied Optimization. Springer, 2005, p. 40.

[28] W. Spears et al. “An overview of evolutionary computation”. In: *Machine Learning: ECML-93*. Ed. by P. Brazdil. Vol. 667. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1993, pp. 442–459.

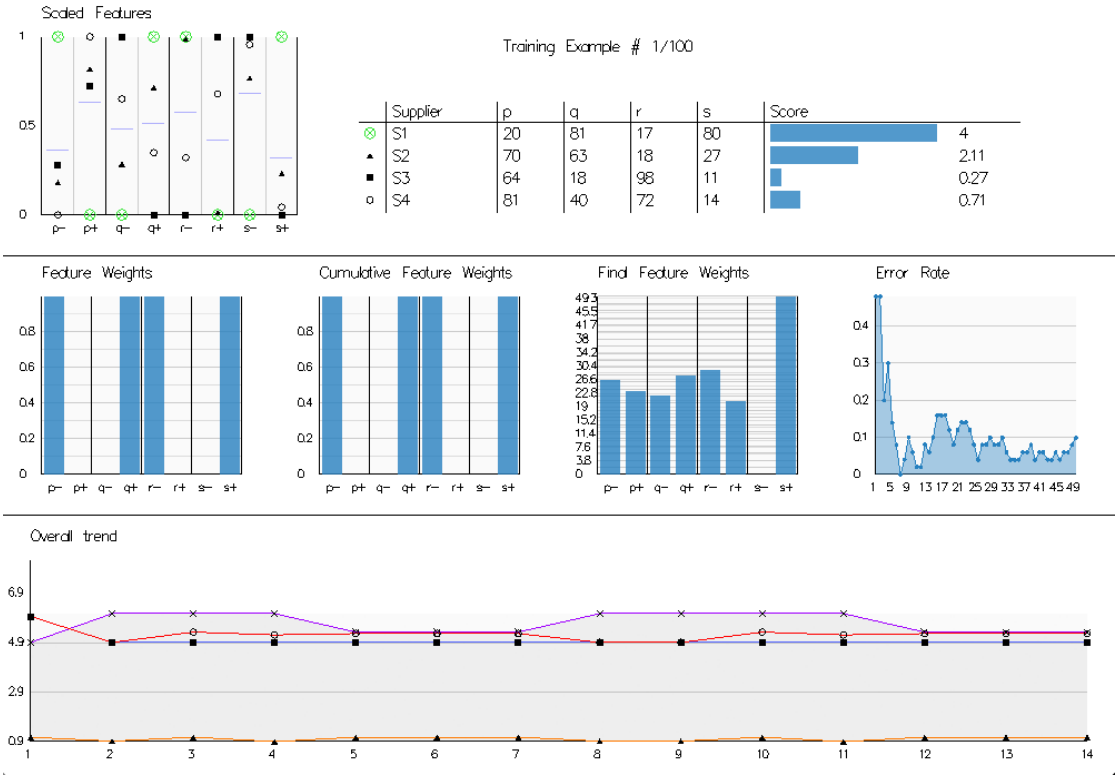
[29] J. Taylor. “Decision Management Systems: A Practical Guide to Using Business Rules and Predictive Analytics”. In: IBM Press. Prentice Hall, 2011, pp. 49–53.

[30] T. E. Vollman. “Manufacturing Planning and Control for Supply Chain Management”. In: McGraw-Hill, 1997. Chap. 1.

[31] J.-B. Waldner. “Principles of Computer Integrated Manufacturing”. In: John Wiley and Sons, 1992, pp. 128–132.

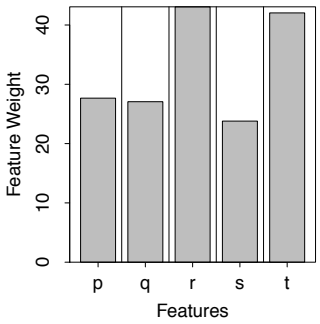
[32] I. Watson and F. Marir. “Case-based reasoning: A review”. In: *The Knowledge Engineering Review* 9.04 (1994), pp. 327–354.

A Prototype solution - user interface

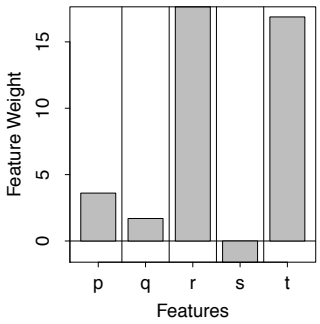


B Test results using artificial data

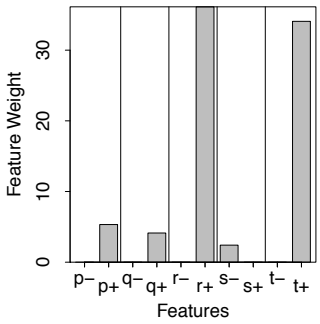
These figures show the weights and error rates using an artificially generated dataset with 5 features. When generating the training-set, the sum of 2 randomly selected features was used to selected the best alternative.



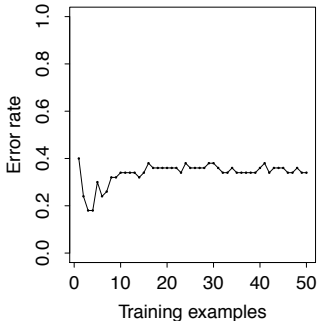
(a) *Weights - Algorithm A*



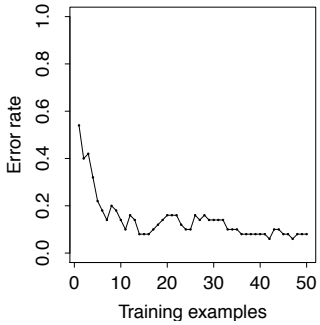
(b) *Weights - Algorithm B*



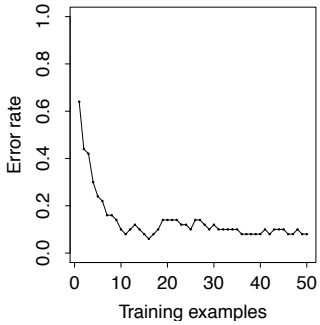
(c) *Weights - Algorithm C*



(d) *Error-rate - Algorithm A*

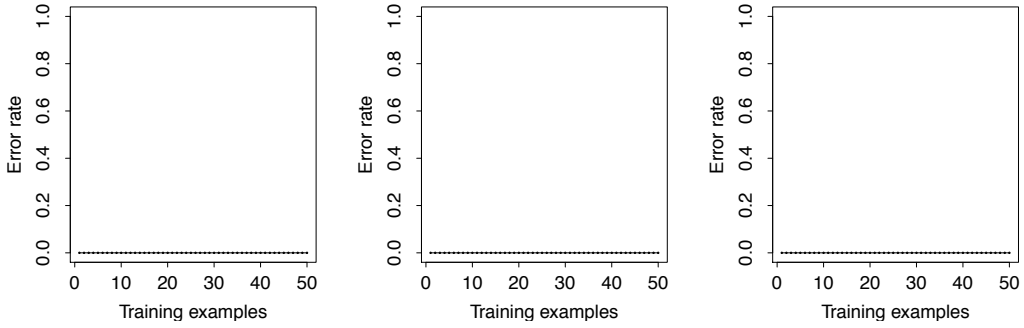


(e) *Error-rate - Algorithm B*



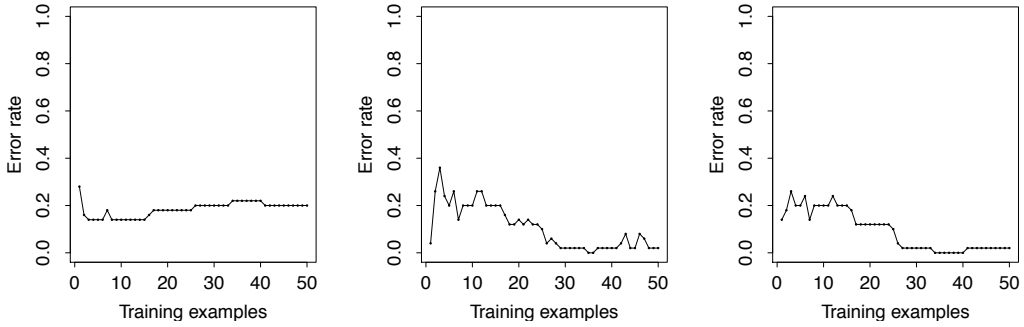
(f) *Error-rate - Algorithm C*

C Output analysis charts



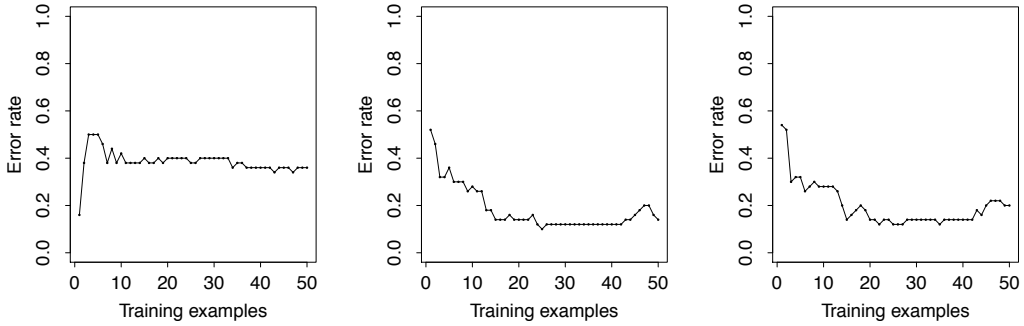
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure C.1: *Artificial data,* $n = 1$*,* $m = 50$



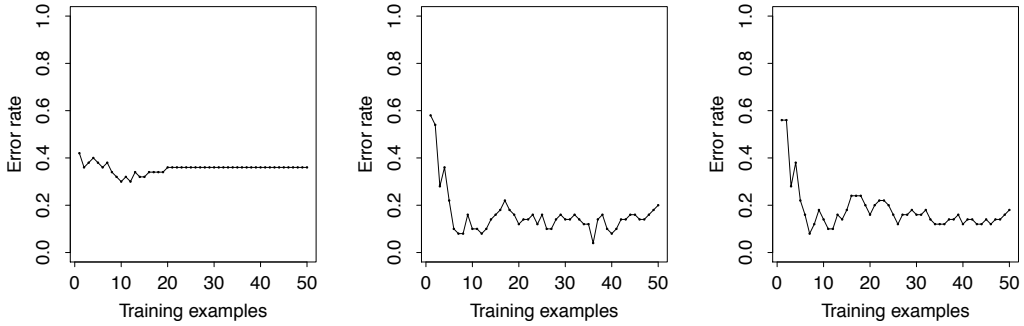
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure C.2: *Artificial data,* $n = 2$*,* $m = 50$



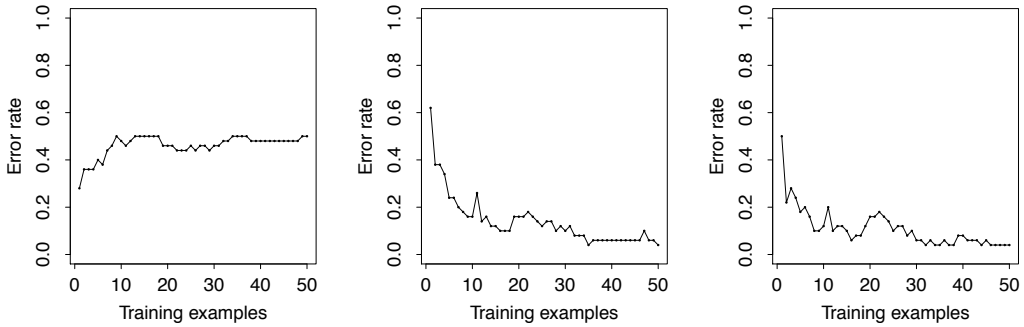
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure C.3: *Artificial data,* $n = 3$*,* $m = 50$



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

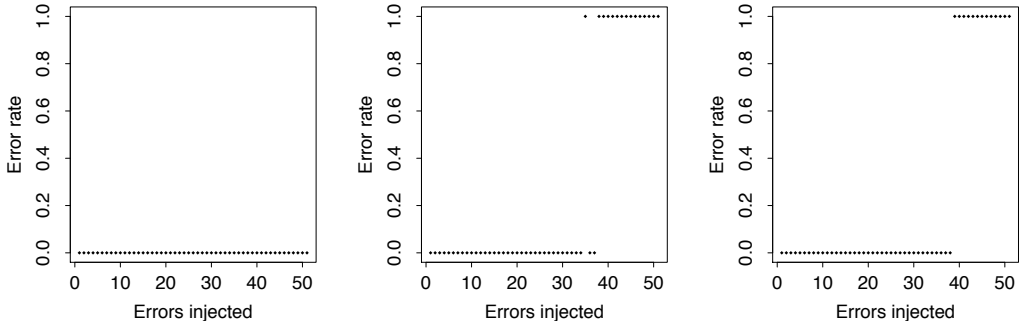
Figure C.4: *Artificial data,* $n = 4$, $m = 50$



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

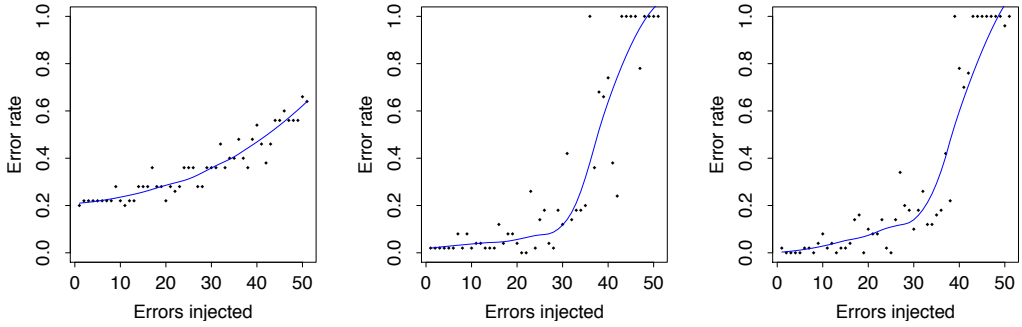
Figure C.5: *Artificial data,* $n = 5$, $m = 50$

D Sensitivity analysis charts



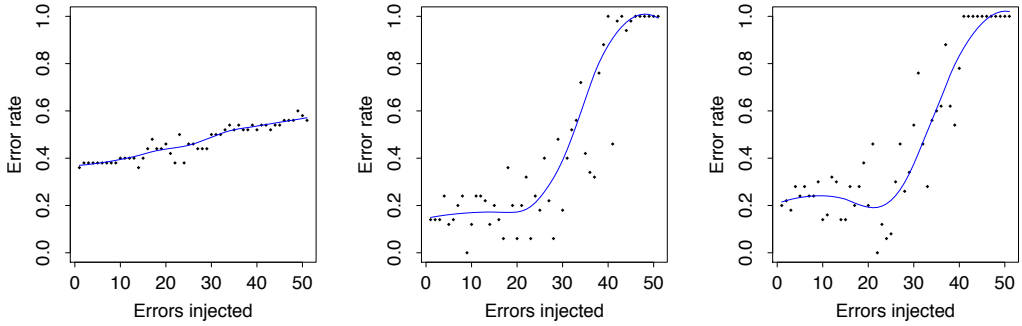
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure D.1: *Artificial data,* $n = 1$*,* $m = 50$



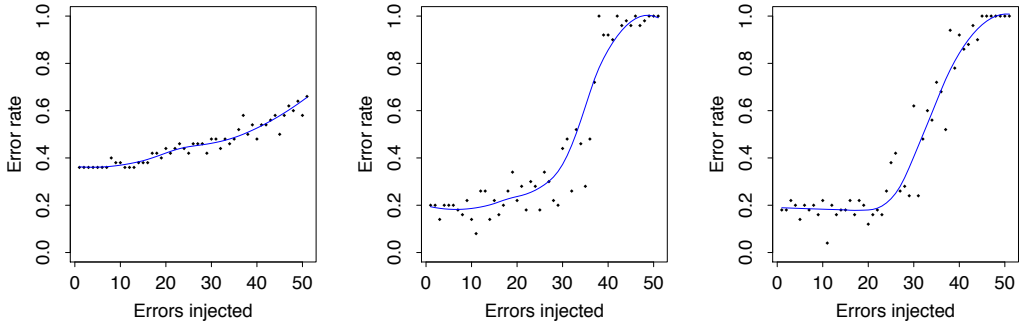
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure D.2: *Artificial data,* $n = 2$*,* $m = 50$



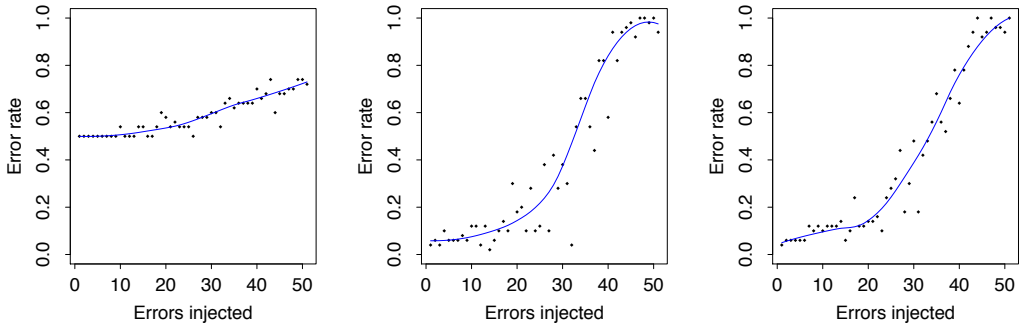
(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure D.3: *Artificial data,* $n = 3$*,* $m = 50$



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure D.4: *Artificial data,* $n = 4$, $m = 50$



(a) *Algorithm A* (b) *Algorithm B* (c) *Algorithm C*

Figure D.5: *Artificial data,* $n = 5$, $m = 50$

E Haskell prototype documentation

E.1 MachineLearning.Requisitions.Requisitions

```
module MachineLearning.Requisitions.Requisitions (
    FeatureSet, FeatureWeights, TrainingExample, TrainingSet, RankFunction,
    AlgorithmConfig(AlgorithmConfig,
                    getTrainingFn,
                    getRankingFn,
                    getPreScaleFn,
                    getPostScaleFn,
                    getFlipPolarities,
                    getFeatureGroupSize),
    algorithmConfigs, getAlgorithmConfig, featureLabels, scaleFeatures,
    featureWeightsDAvg, featureWeightsSum, rank, pickBest, errorRate,
    flipPolarity, flipPolarity', addPolarizedFeatures,
    addPolarizedFeatures', rank', rank''
  ) where
```

E.1.1 Type Synonyms

```
type FeatureSet = Matrix Double
```

A block-matrix with supplier feature values (features across columns).

```
type FeatureWeights = Matrix Double
```

A row-vector of feature weights.

```
type TrainingExample = (FeatureSet, Int)
```

A combination of a feature matrix and picked supplier index.

```
type TrainingSet = [TrainingExample]
```

```
type RankFunction
```

`=`	`FeatureSet`	Set of supplier features (matrix).
`->`	`FeatureWeights`	Feature weights (row vector).
`->`	`Matrix Double`	Supplier scores (column vector).

E.1.2 Configurations

```
data AlgorithmConfig
```

```
    = AlgorithmConfig
    getTrainingFn ::  TrainingSet -> FeatureWeights
    getRankingFn ::  RankFunction
    getPreScaleFn ::  FeatureSet -> FeatureSet
    getPostScaleFn ::  FeatureSet -> FeatureSet
    getFlipPolarities ::  Bool
    getFeatureGroupSize ::  Float
```

```
algorithmConfigs :: [(Char, AlgorithmConfig)]
```

```
getAlgorithmConfig :: Char -> AlgorithmConfig
```

```
featureLabels :: Float -> [String]
```

E.1.3 Functions

`scaleFeatures :: FeatureSet -> FeatureSet`

Scales the columns of the given matrix such that the values are all within the range <0, 1>.

`featureWeightsDAvg :: TrainingSet -> FeatureWeights`

Maps distanceToAvg function over all training examples and sums the results.

`featureWeightsSum :: TrainingSet -> FeatureWeights`

Alternative method of getting feature weights. Instead of distance to average feature value, it uses the absolute feature values as the weights.

`rank :: RankFunction`

Ranks the supplier's feature set, returning a column-vector of scores.

```
r = features * (weights T)
```

`pickBest :: RankFunction -> FeatureSet -> FeatureWeights -> Int`

Returns the index of the highest-ranked supplier.

```
errorRate :: RankFunction
             -> TrainingSet -> FeatureWeights -> Double
```

Calculates the errorrate of the supplied weights when using them on a training set. Lower value is better.

`flipPolarity :: FeatureSet -> FeatureSet`

Flip the polarity of the entire feature set (negating all values)

`flipPolarity' :: [Bool] -> FeatureSet -> FeatureSet`

Flips the polarity of the columns in the supplied boolean list.

`addPolarizedFeatures :: FeatureSet -> FeatureSet`

Add polarized features to feature set. p -> p-, pAverage, p+

`addPolarizedFeatures' :: FeatureSet -> FeatureSet`

Add polarized features to feature set. p -> p-, p+

`rank' :: RankFunction`

Rank using only highest weight of the polarized features.

`rank" :: RankFunction`

Rank using only highest weight of the + and - polarities. The resulting weight used is the (absolute) difference between + and -.

E.2 MachineLearning.Requisitions.IO

```
module MachineLearning.Requisitions.IO (
    trainingSetFromCSV,  trainingSetFromCSV',  trainingSetToCSV,
    trainingSetToCSV'
  ) where
```

E.2.1 Read

`trainingSetFromCSV`

`::`	`String`	CSV input string.
`->`	`Bool`	Flip polarities
`->`	`(FeatureSet -> FeatureSet)`	Transformation function.
`->`	`IO (TrainingSet, Matrix String)`	

Reads a set of training examples from a CSV string.

All lines will be grouped into matrices by their first value, REQ_ID. At the moment the first occurrence of a new REQ_ID is assumed to be the picked supplier.

Expected input format:

```
REQ_ID, SUPPLIER_ID, x1, x2, x3
REQ_ID, SUPPLIER_ID, x1, x2, x3
```

`trainingSetFromCSV'`

`::`	`FilePath`	The csv file to read.
`->`	`Bool`	Flip polarities
`->`	`(FeatureSet -> FeatureSet)`	Transformation function.
`->`	`IO (TrainingSet, Matrix String)`	

Reads a set of training examples from a CSV file.

E.2.2 Write

`trainingSetToCSV :: TrainingSet -> String`

Writes a training set to a comma separated values string. Well..semicolon separated really...

`trainingSetToCSV' :: FilePath -> TrainingSet -> IO ()`

Writes a training set to a csv file.

E.3 MachineLearning.Requisitions.Random

```
module MachineLearning.Requisitions.Random (
    defaultFMin, defaultFMax, generateSet, generateSet', generateFeatures,
    generateFeatures', generatePicks, insertErrors, randRangeExcept,
    randOneOf, uniqueRandList
  ) where
```

E.3.1 Default values

`defaultFMin :: Int`

The default minimum feature value.

`defaultFMax :: Int`

The default maximum feature value.

E.3.2 Generating new sets

`generateSet`

`::`	`Int`	The number of orders to generate.
`->`	`Int`	The number of suppliers per order.
`->`	`Int`	The number of features per supplier.
`->`	`[Int]`	The feature to base picking the best supplier on.
`->`	`IO TrainingSet`	

Generates a random training set.

`generateSet'`

`::`	`Int`	The number of orders to generate.
`->`	`Int`	The number of suppliers per order.
`->`	`Int`	The number of features per supplier.
`->`	`Int`	Minimal feature value.
`->`	`Int`	Maximum feature value.
`->`	`[Int]`	The feature to base picking the best supplier on.
`->`	`IO TrainingSet`	

Generates a random training set.

`generateFeatures`

`::`	`Int`	The number of suppliers to generate.
`->`	`Int`	The number of features per supplier.
`->`	`IO FeatureSet`	

Generates a single random feature set.

`generateFeatures'`

`::`	`Int`	The number of suppliers to generate.
`->`	`Int`	The number of features per supplier.
`->`	`Int`	Minimal feature value.
`->`	`Int`	Maximum feature value.
`->`	`IO FeatureSet`	

Generates a single random feature set.

`generatePicks`

`::`	`[FeatureSet]`	The features to compare.
`->`	`[Int]`	The feature column indices to base the picking on.
`->`	`[Int]`	

Generates a list of picked supplier indices, based on the highest value in the given columns.

E.3.3 Inserting errors

`insertErrors`

`::`	`TrainingSet`	The training set to modify.
`->`	`Int`	Number of errors to introduce.
`->`	`IO TrainingSet`	

Changes the picked value in a given number of random training examples to a random value, where the new value /= original value.

E.3.4 Helpers

`randRangeExcept :: Int -> Int -> [Int] -> IO Int`

Generates a random integer in the supplied range. It ensures that the generated number is not an element in the supplied exclude list.

```
randOneOf :: [a] -> IO a
```

Picks a random element from the supplied list.

```
uniqueRandList
    ::  Int         The lower range bound.
    ->  Int         The upper range bound.
    ->  Int         The number of numbers to generate.
    ->  IO [Int]
```

Generates a list of unique random numbers in the supplied range.

E.4 MachineLearning.Maths.Matrix

```
module MachineLearning.Maths.Matrix (
    Matrix(M, rows), (<+>), (<|>), (<->), (<\>), mIdentity, mEmpty,
    mInfinite, numRows, numCols, numElements, size, isEmpty, cols,
    elements, element, row, col, (!!!), (!-!), (!|!), mFirst, (+-+),
    (+|+), takeR, dropR, takeC, dropC, repeatR, repeatC, mTranspose,
    mMap, rMap, cMap, mZip, (!+), (!-), (!*), (!/), mMul, chunksC'
  ) where
```

A simple matrix module which supports basic matrix transformations such as by-element and scalar arithmetic operations and matrix multiplication.

E.4.1 Constructors

```
data Matrix a
    =               M
    rows ::  [[a]]

instance Functor Matrix
instance Eq a => Eq (Matrix a)

instance Fractional a => Fractional (Matrix a)
```

By-element fractional operations (/).

```
instance Num a => Num (Matrix a)
```

By-element numerical operations (+), (-), (*).

```
instance Read a => Read (Matrix a)
instance Show a => Show (Matrix a)

(<+>) :: (Int, Int) -> [a] -> Matrix a
```

Block-matrix constructor. The first value in the tuple is the number of rows, the second the number of columns.

```
>>> (2, 3) <+> [1..]
[1, 2, 3]
[4, 5, 6]
```

```
(<|>) :: Int -> [a] -> Matrix a
```

Column vector constructor.

```
>>> 2 <|> [1..]
[1]
[2]
```

`(<->) :: Int -> [a] -> Matrix a`

Row vector constructor.

```
>>> 3 <-> [1..]
[1, 2, 3]
```

`(<\>) :: (Int, Int) -> (a, a) -> Matrix a`

Identity matrix constructor.

```
>>> (2, 3) <\> ('x', 'y')
['y', 'x', 'x']
['x', 'y', 'x']
```

`mIdentity :: Num a => Int -> Matrix a`

Creates a square numeric identity matrix

```
>>> mIdentity 2 :: Matrix Float
[1.0, 0.0]
[0.0, 1.0]
```

`mEmpty :: Matrix a`

Creates a matrix with 0 elements.

`mInfinite :: a -> Matrix a`

Creates a matrix with an infinite size.

E.4.2 Matrix Properties

`numRows :: Matrix a -> Int`

Gets the number of rows in a matrix.

`numCols :: Matrix a -> Int`

Gets the number of columns in a matrix.

`numElements :: Matrix a -> Int`

Gets the number of cells in a matrix.

`size :: Matrix a -> (Int, Int)`

Gets the size of a matrix.

`isEmpty :: Eq a => Matrix a -> Bool`

Tests if the matrix is empty.

E.4.3 Sub-matrices

`cols :: Matrix a -> [[a]]`

Gets a list of columns of a matrix.

`elements :: Matrix a -> [a]`

Gets a single list with all elements of a matrix.

`element :: Matrix a -> (Int, Int) -> a`

Gets the element in row r, column c.

`row :: Matrix a -> Int -> Matrix a`

Gets the i-th row of a matrix.

`col :: Matrix a -> Int -> Matrix a`

Gets the i-th column of a matrix.

`(!!!) :: Matrix a -> (Int, Int) -> a`

Alternative to the element function.

`(!-!) :: Matrix a -> Int -> Matrix a`

Alternative to the row function.

`(!|!) :: Matrix a -> Int -> Matrix a`

Alternative to the col function.

`mFirst :: Matrix a -> a`

Returns the upper-left value of the matrix.

`(+-+) :: Matrix a -> Matrix a -> Matrix a`

Appends matrix b to matrix a as new rows.

`(+|+) :: Matrix a -> Matrix a -> Matrix a`

Appends matrix b to matrix a as new columns.

`takeR :: Int -> Matrix a -> Matrix a`

Takes n rows from a matrix.

`dropR :: Int -> Matrix a -> Matrix a`

Drops n rows from a matrix.

`takeC :: Int -> Matrix a -> Matrix a`

Takes n columns from a matrix.

`dropC :: Int -> Matrix a -> Matrix a`

Drops n rows from a matrix.

`repeatR :: Matrix a -> Matrix a`

Repeats the rows of a matrix

`repeatC :: Matrix a -> Matrix a`

Repeats the columns of a matrix

E.4.4 Matrix Transformations

`mTranspose :: Matrix a -> Matrix a`

Transpose the matrix.

`mMap :: (a -> b) -> Matrix a -> Matrix b`

Maps a function over each element in the matrix.

`rMap :: ([a] -> [b]) -> Matrix a -> Matrix b`

Maps a function over each row in the matrix.

`cMap :: ([a] -> [b]) -> Matrix a -> Matrix b`

Maps a function over each column in the matrix.

`mZip :: (a -> b -> c) -> Matrix a -> Matrix b -> Matrix c`

Zips two matrices with a given function f.

E.4.5 Arithmetic operations

`(!+) :: Num a => Matrix a -> a -> Matrix a`

Scalar addition.

`(!-) :: Num a => Matrix a -> a -> Matrix a`

Scalar subtraction.

`(!*) :: Num a => Matrix a -> a -> Matrix a`

Scalar multiplication.

`(!/) :: Fractional a => Matrix a -> a -> Matrix a`

Scalar division.

`mMul :: Num a => Matrix a -> Matrix a -> Matrix a`

Matrix multiplication.

E.4.6 Miscellaneous functions

`chunksC' :: Matrix a -> Int -> [Matrix a]`

Chops a matrix into column chunks of the given length.

```
>>> chunksC' ((2, 6) <+> [1..]) 3
[[1,2,3]
 [7,8,9],
 [4,5,6]
 [10,11,12]]
```

E.5 MachineLearning.Maths.Statistics

```
module MachineLearning.Maths.Statistics (
    rSum, cSum, mSum, rMean, cMean, mMean, quantile, median, quartiles,
    fiveNum, rStdDev, cStdDev, rNorm, cNorm, mse, rmse, corr, covariance
  ) where
```

A collection of statistical operations on matrices.

E.5.1 Sum and mean values

`rSum :: Num a => Matrix a -> Matrix a`

Sums up all rows and returns the sums as a column vector

`cSum :: Num a => Matrix a -> Matrix a`

Sums up all columns and returns the sums as a row vector

`mSum :: Num a => Matrix a -> a`

Sums up all values in a matrix.

`rMean :: Floating a => Matrix a -> Matrix a`

Calculates the mean value of each row in a matrix.

`cMean :: Floating a => Matrix a -> Matrix a`

Calculates the mean value of each column in a matrix.

`mMean :: Floating a => Matrix a -> a`

Calculates the mean over of values in a matrix.

E.5.2 Quantile

`quantile :: RealFrac a => a -> [a] -> a`

Gets a given quantile from a list of values.

```
>>> quantile 0.25 [1,2,3,4,5]
2.0

>>> quantile 0.75 [1,2,3,4,5]
4.0
```

`median :: RealFrac a => [a] -> a`

Gets the median from a list of values.

`quartiles :: RealFrac a => [a] -> (a, a, a)`

Gets three values dividing a list into quartiles. (Q1, median, Q3).

```
>>> quartiles [1,2,3,4]
(1.5,2.5,3.5)
```

`fiveNum :: RealFrac a => [a] -> (a, a, a, a, a)`

Gets the five-number summary from a list. (min, Q1, median, Q3, max).

```
>>> fiveNum [1,2,3,4]
(1.0,1.5,2.5,3.5,4.0)
```

E.5.3 Standard Deviation and Normalization

`rStdDev :: Floating a => Matrix a -> Matrix a`

Returns a column vector with the standard deviation of each row in a matrix.

`cStdDev :: Floating a => Matrix a -> Matrix a`

Returns a row vector with the standard deviation over all rows.

`rNorm :: Floating a => Matrix a -> (Matrix a, Matrix a, Matrix a)`

Normalizes each row in a matrix.

Returns a triple: (Matrix normalized over columns, means column vector, std. dev. column vector).

`cNorm :: Floating a => Matrix a -> (Matrix a, Matrix a, Matrix a)`

Normalizes each column in a matrix.

E.5.4 Mean Squared Error

`mse :: Floating a => Matrix a -> Matrix a -> a`

Mean squared error.

```
mse(x,y) = 1/2n * sum( (x(i) - y(i)) ^ 2)
```

`rmse :: Floating a => Matrix a -> Matrix a -> a`

Root mean squared error.

E.5.5 Correlations

`corr :: Floating a => Matrix a -> Int -> Int -> a`

Pearson's correlation between columns x and y in matrix m.

```
corr(X,Y) = cov(X,Y) / (stdDev(X) stdDev(Y))
```

`covariance :: Floating a => Matrix a -> Matrix a -> a`

Covariance between two column-vectors.

```
q(jk) = 1 / (n - 1) * sum (x(ij) - mean( x(j) )) * ( x(ik) - mean( x(j) ) )
```