Deloitte

KNOWLEDGE DISCOVERY FOR DELOITTE INVISION WEBSERVICES

by

FRANK W. VAN DEN NIEUWBOER

Department of Computing Science
University of Groningen

A Research Thesis submitted in partial fulfillment
of the requirements for the Master degree in
Computing Science, September 27^{th}, 2006

Supervisors :

MR. J. GROENEWOLD	Deloitte, Enterprise Risk Services
DR. R. SMEDINGA	University of Groningen, Department of Computing Science
DR. M. BIEHL	University of Groningen, Department of Computing Science

RuG

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER i

ABSTRACT

In many systems log information concerning event timing is available. Information about the performance of the system is concealed within this log information. This thesis describes a method to extract the performance information, using data mining techniques. In order to extract performance information, we need to detect at which moment a system reacts slow.

The first technique applied to detect outliers is anomaly detection. We make use of an algorithm called Smartsifter, which uses Gaussian Mixture Models to maintain models of a distribution of the input data. Each time a new data point is added, the model is updated. A score is calculated out of the change of the model, if this score is above some threshold, an outlier is reported. The next step is to classify the concurrent situations. Using Relevance Learning Vector Quantization we create so-called prototype vectors and relevance vectors, from which we can deduct which events are important for which class membership. We implemented this framework in a prototype application, and set it to work at a 40 gigabyte large database of time log information, which was extracted from the Deloitte INVision framework. From the results of the prototype application we can conclude that we can implement a performance measure framework using data mining techniques, which extracts system performance information from log timing information. For Deloitte INVision, a better timeout procedure can be developped using this technique.

CHAPTER ii

PREFACE

With the introduction of larger and faster information systems, more information is processed within the same time. This leads to an enormous amount of data containing a huge amount of knowledge about the systems. Information retrieval from large data stores is considered to be the field of data mining and knowledge discovery, which is becoming increasingly popular due to its great advantages such as better insight. For Deloitte, data mining techniques are used to determine performance information for a web-based application called Deloitte INVision. With the help of information retrieval and statistical algorithms, an environment has been setup which is capable of delivering the performance information about the system, in order to improve error detection.

This thesis concludes the master research that has been carried out at the Enterprise Risk Services (ERS) department of Deloitte, under the supervision of mr. J. Groenewold (Deloitte). The research has been carried out over a period of 7 months, from March 2006 until the end of September 2006. The thesis first describes how the research took place, what kind of systems we are dealing with, and how the data is retrieved such a way that no company policies are neglected. After sketching this, using the appropriate statistical algorithms, a theoretical solution for the problem will be introduced, and with the help of a prototype, a solution will be presented and validated.

This thesis mostly aims at scientific readers, who are interested in the field of performance information measures and retrieval out of large information systems. Although the presented solution is unique for Deloitte, similar problems could be tackled using a somewhat similar solution. For this purpose the ideas are introduced as generic as possible to encourage further research.

Acknowledgements

Several people have contributed to the successful completion of this research project. The author would like to thank the supervisor from Deloitte, J. Groenewold (Deloitte) for his useful comments and support. Next the author would like to thank Dr. R. Smedinga (RuG) and Dr. M. Biehl (RuG), both supervisors from the university, for their advice during the project. Lastly, the following people have contributed to this research project: Prof Dr. M. de Rijke (UvA), Dr. K. Yamanishi (NEC), Drs. W. Diele (Deloitte), Drs J. Jongejan (RuG), Dr. J. Peij (Simon Fraser University), Ir. H. Braam (Deloitte), Drs. A. Ghosh (RuG), Drs. P. Schneider (RuG)

Frank van den Nieuwboer, Groningen, September 27^{th}, 2006

CHAPTER iii

MANAGEMENT SUMMARY

This research thesis contains the research results of a 7 month research trail at the Enterprise Risk Services department of Deloitte. Within this department a web based tool called Deloitte INVision is used and developed as a framework for audits, benchmarks and surveys.

The Deloitte INVision framework serves up to 6000 users each day, and the number of supported users is growing. In order to make predictions about the system behavior when the number of users (and inherently the rate of concurrency) is increased, a research has been set up to determine performance information about the system. Because the information within the system is confidential, a log-mechanism has been implemented in the Deloitte INVision engine, which is the core of the system. Using data mining techniques this research tries to extract information from the log information.

In an infinite series of events, how can we determine which events are outliers, and how can we predict combinations of concurrent processed events which cause exceptional processing times, using log-based event timing information.

The main research question contains the performance questions and concentrates on outlier detection and database and locking problems, which might be solved by the application programmers. To support the main research question three research subquestions are introduced:

- Q1 : For each transaction, how can determine which points are outliers using a dynamic measure to approach the *normal transaction time* in some time frame
- Q2 : What is the relation between events processed concurrently and events processed sequential
- Q3 : How can we find out if an anomalous event duration within a concurrent set of events can be due to resource conflicts or other problems.

In order to solve the research subquestions and the main research question, various data mining techniques were explored.

The presented solution for the main research problem should be as generic as possible, so a flexible solution of three components was chosen. First concurrency is detected, by comparing event starting and ending times. Secondly an anomaly detection algorithm is used to detect anomalous points. In order to provide a dynamic solution, the

anomaly detection algorithm of Yamanishi ([Yam00]) was chosen, which uses Gaussian Mixture Models to maintain models of the distributions of the data input points. Lastly a classification algorithm was used to classify the concurrent states[1]. Relevance Learning Vector Quantization ([BHSvT]) seemed to do the trick.

The solution which was introduced has been implemented in a prototype application, and has been tested on two large real datasets from the Deloitte INVision framework. The results show the concurrency detection, the outlier detection and classification algorithm operate well, although the results of classification do not exactly satisfy the third sub question. The prototype uses a small amount memory and computes the results fast.

The tests of the prototype showed that the results of a large dataset does not differ a lot from the results of a small dataset.

Recapitulating, the solution presented in this research paper fits well to detect concurrency, and to determine outliers within the input data. The classification works well, although some more data mining techniques should be applied to present more detailed performance information. For Deloitte the prototype is very useful because it delivers more detailed error information.

[1] A state is a concurrent situation in which different function calls are running at the same time

CHAPTER 1

INTRODUCTION

"Computers are useless. They can only give you answers." *Pablo Picasso*

This chapter introduces the system environment of Deloitte INVision, the problem context, and gives some background information

1.1. Background

The need for better and more detailed performance information concerning company critical applications is growing extensively, while determining this information is a very complex task.

For about six months, with the co-operation of others, a solution has been successfully completed for a specific case of this problem in the context of the Enterprise Risk Services department of Deloitte in the Netherlands. Though motivation, goal and focus lay on different aspects for those involved, this document describes the communication and research accomplishments of the project, and suggests ideas for further research and similar projects.

1.1.1. Accountancy

The profession of accountancy exists since the early days of human agriculture and civilization, when the need to maintain accurate records of the quantities and relative values of agricultural products first arose. Since the 17th century, the science of accounting was highly appreciated. Accountancy consists of the measurement, disclosure or provision of assurance about information that helps decision makers with their resource allocation decisions. Financial accounting is one branch of accounting, and is involved in the processes which record, summarize, classify and communicate the financial information about a business. Audit is a related separate discipline, which involves the process whereby an independent auditor examines an organization's financial statements and accounting records in order to express an opinion about the truth, fairness and adherence to general accounting principles of all the materials. A sub-branch of the principle is risk-auditing, in which the auditor tasks are mainly focused on the enterprise risks.

1.1.2. Sarbanes-Oxley

Since the Wall-street scandals[1] in 2002, the American Congress has established the "Sarbanes-Oxley" Act. This Act covers issues such as *establishing a public company accounting oversight board, auditor independence, corporate responsibility* and *enhanced financial disclosure*. It was designed to improve the out-dated legislative audit requirements, and is considered one of the most significant changes to the United States securities law since the New Deal[2] in the 1930s. The act gives additional powers and responsibilities to the US Securities and Exchange Commission.

1.1.3. Deloitte INVision

To comply to this Act, Deloitte has envised an web based application platform called Deloitte INVision, which provides support for benchmarks, performance measures, surveys and audits. It supports accountants and auditors during their auditing tasks. Especially in the risk consulting and risk auditing Deloitte INVision is useful compliance tool. The Sarbanes-Oxely act devises the accountants to deliver risk management approval certificates to companies in order to make them conform to the law, and Deloitte INVision is a perfect tool to support them..

The power of Deloitte INVision, is that it is capable to register a complete audit trail, or the audit actions within a industrial process. Each action within the audit trail can be assigned as an activity which has to be accorded in the tool. This way the complete industrial process (with all activities) can be elaborated in the tool, and all activities can be accredited by responsible persons. This capability makes Deloitte INVision very useful for auditors, who model the complete audit trail in Deloitte INVision, and then perform a verified audit for a customer. Deloitte INVision is used within Deloitte itself, but also sold to customers as an internal audit control.

Deloitte INVision is the playground and basis of this master research project. Deloitte INVision consists of a 4-tier model: A Database component, a Knowledge component, a Web component for on line access, and a Browser at client side. The system is based on standard Microsoft Technology and the hardware

[1] Major corporate and accounting scandals including those affecting Enron, Tyco International, and WorldCom

[2] The New Deal : the name given to the series of programs implemented between 1933-37 under President Franklin D. Roosevelt with the goal of relief, recovery and reform of the United States economy during the Great Depression

is outsourced to a third party. There are multiple servers running the Deloitte INVision framework in a distributed environment.

1.2. Goal

An important question for Deloitte INVision administrators is how the system will react to an increase of users. How does the relation between the number of users and the hardware requirements look like. Does twice the amount of hardware also mean twice the amount of supportable users? Applying data mining techniques on the system log data might give us more insights on how this performance measures look like. Also, at the moment, whenever a function call takes longer than three seconds, it is reported as an error. But could we do better, and provide better error reports using a technique to dynamically find out the maximum duration of a function call. The results of this research might be interesting for application programmers and system administrators.

1.3. Problem Environment

As described in the section 1.1, Deloitte INVision is a framework which is based on a 4-Tier model, consisting of Microsoft software technology. The system architecture consists of a Microsoft SQL Server as data store, an Application Server, a Microsoft Internet Information Server, and an Internet Explorer browser at the client-side; the 4-Tier model suits well for an application for Audit purposes, the data store is the heart of the application.

In the production environment the system is rolled out using multiple servers, and a large data store for the database management system. Currently it serves approximately 6000 users during their daily (audit) tasks. A failure of the system will result in many users not being able to do their activities, which in its turn results in company dis-performance. This makes the system a critical system.

The system has been developed and built on top of native Microsoft Windows Technology, the Deloitte INVision engine, which is the main component, has a C++ implementation and the database is of the type OLE-DB[3]. When the system is started the Deloitte INVision engine is loaded as a Dynamic Link Library in the Web-server.

To provide dynamic behavior in the content delivered to the users, the Inter-

[3]OLE-DB : Object Linking and Embedding Database

net Information Server uses ASP[4] technology to handle the web requests of the users. The dynamic ASP pages call functions on the COM interfaces of the Deloitte INVision Engine running at the application server. A screen shot of how a web page in the Deloitte INVision environment looks like is shown in figure 1.1. The left part of the screen contains a tree in which several items, which are called dossiers, are shown. When a user clicks a button, or adds text to the text a button, logic is carried out, and transactions are executed.

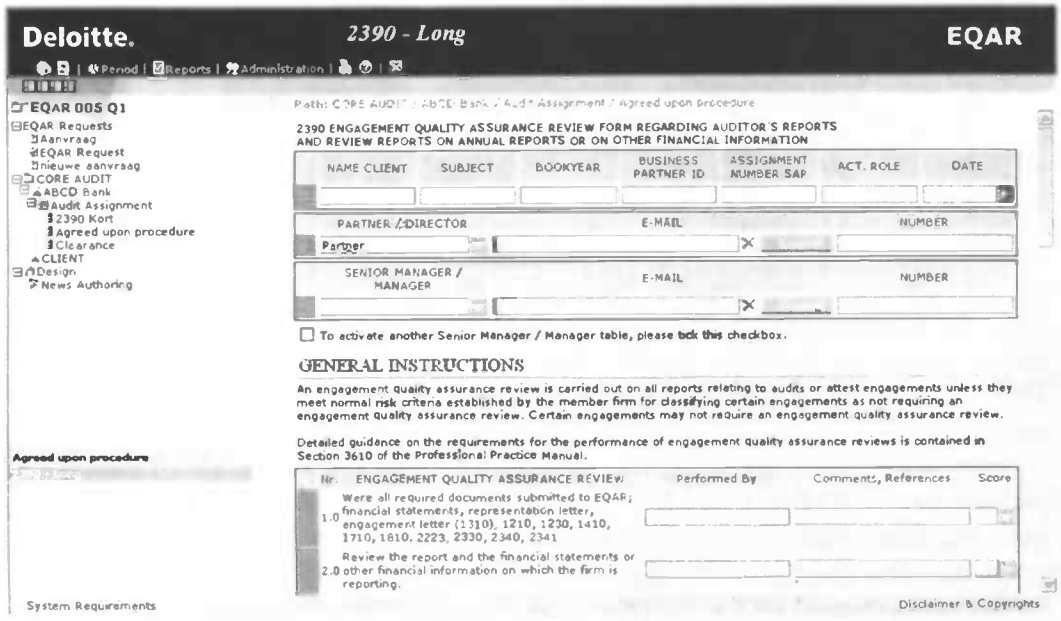

Figure 1.1.: A screen-shot of a solution in the Deloitte INVision framework

The Deloitte INVision engine has interfaces to handle different types of requests, runs the logic required, and accommodates the specific queries (or call stored procedures) to the data store. The Deloitte INVision engine has been set-up in such a way that it is able to handle multiple instances of the framework at once (an instance of the framework is called a ***Solution***). For each solution a so-called ***solution-object*** is created in memory. These objects deal with all processing concerning a specific solution. A solution-object of a solution is able to create ***transaction-objects*** which are able to handle different actions on the data of the solution.

Each solution in the Deloitte INVision framework is built using a Knowledge

[4] Active Server Pages

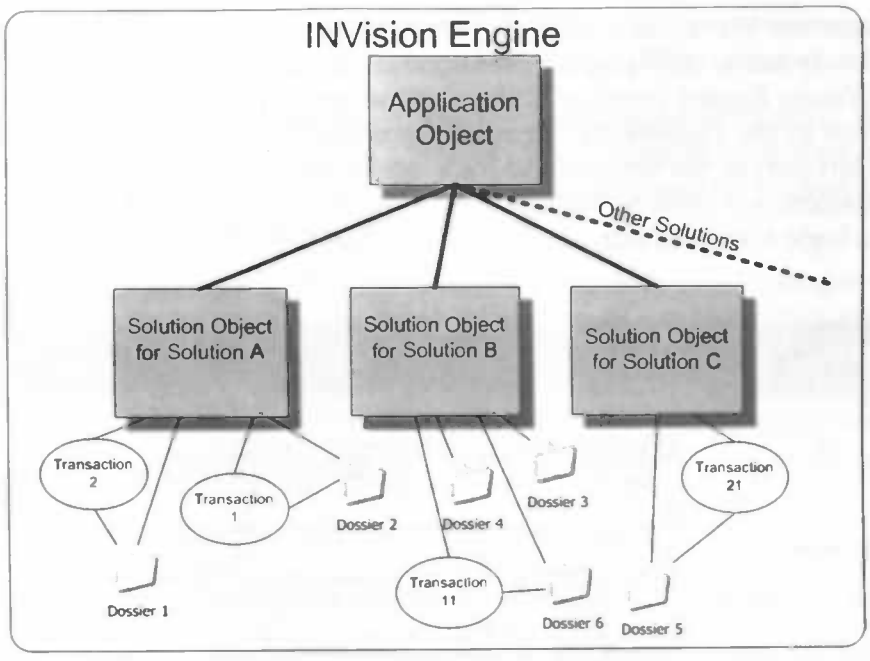

Figure 1.2.: The structure of the engine: Dossier, Application, Solution and Transaction objects

Specification Tool (KST). The KST uses XML[5] to define so-called ***packs***. These packs are **blueprints** for the run-time pages which can be created in a solution within the Deloitte INVision framework. These runtime pages are called *Dossiers*, and consist of the fields which were defined in the packs, combined with the data which was added by the users during runtime. Each pack consists of a set of fields, which are uniquely identified by their ***KST-ID***. In run-time one or more ***dossiers*** can be created using the blueprint which was specified in the packs. Each solution has one or more packs as blueprint for the dossiers. So packs are created during *construction time*, and dossiers during *run-time*.

When a transaction-object of a specific solution-object needs a specific dossier, it asks the solution-object for the dossier. The solution-object returns the specific ***dossier-object*** from memory, or creates a new dossier-object in memory in which the dossier is loaded from the database. The solution-object keeps track of open dossier-objects. When consumed memory of a solution becomes too large, the dossier-objects which have not been used for a while are released.

Another important task of the engine is to manage the user authentication, and

[5]Extensible Markup Language

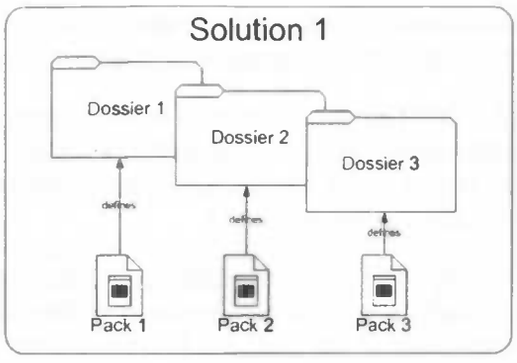

Figure 1.3.: The structure: Solution, Dossier, Pack

access to specific solutions and dossiers.

The data store, which is modeled by a Microsoft SQL Server Database Management System contains all the data for each solution in a database. In the Database Management System a ***Meta data Model*** is used to model the solution specific data. The **design and the data** of each solution are incorporated in the meta data model, so the set of database tables is for each solution identical. Each database has many indexes and triggers to support faster database reactions and the possibility to use sophisticated functions at database level.

Every time a function within the Deloitte INVision engine has been invoked, the start time, and the time the function needs to complete is recorded and stored in a special Log-database. There is a difference in function duration when the results can be read directly from memory, and when the results have to be fetched from the database. In the Deloitte INVision engine functions that are executed by other functions are logged separately, in order to prevent huge variations in function durations. This way every *function call* (or *call*) can be traced. This Log-database is the basis for this research.

1.4. Outliers

In the Deloitte INVision framework, but also within the Microsoft SQL server, the web-server and the system hardware, problems may occur which result in incorrect or slow system behavior. The detection of occurrence of these problems is difficult whilst the number of users working with the Deloitte INVision framework is considerable. The monitoring of the current system status can be accomplished in three different areas:

- **User Interaction** : When a user sends a remark about the system being dilatory, system administrators check the current status of the system.
- **Continuous Load Monitoring** : The hardware systems running the Deloitte INVision framework are continuously monitored. If system load is too high or system errors occur, the system is examined by system administrators and eventually can be restarted.
- **Log based Monitoring** : The problem of the previous two strategies is that when a problem occurs, in most cases the cause of the problem is not clear because of the complexity of the Deloitte INVision framework. Log based monitoring uses the system log files to analyze how the current status of the system is and in case of problems, report to the system administrators. Log based Monitoring also points out directly to the cause of the misbehavior.

When problems are detected they can be divided in two categories:

- **Failure** : The state of the system is in such a way the system needs to reconfigure in order to continue to function normally.
- **Resource Conflicts** : The system is in a waiting state or the system is running a source-demanding process which blocks other processes, which results in reduced system throughput.

In case of system failure, the state of the system is intolerable, and the system needs to restart. System failure is most often caused by bad programming and hardware failure. Installing fail-safe hardware and testing the software rigorously in advance are ways to reduce the risk of system failure. Although system failure is solvable in most cases, the system often need to reconfigure or reboot to continue functioning.

In case of resource conflicts, some process is waiting for another process holding a resource, resulting in reduced performance. In most cases when the first process releases the resource the performance increases again. Detecting the cause of these resource conflicts is very difficult and it is hard to do this by load monitoring or user interaction.

In the current environment of the Deloitte INVision framework resource conflicts and system failure can originate in many points of the system. Because the system depends on hardware and software of third parties, only a specific range of problems can be addressed to the Deloitte INVision engine itself.

1. **Network Connection** : Due to congestion or broken network connections clients might not be able to receive or transfer the information.
2. **Web server Caching** : The web server and the web browser use caching in order to retrieve the information which the user requested faster, but these caches may present obsolete information.
3. **Hardware Caching** : Caching in the system hardware results sometimes in fast completion times, but could also present long completion times due to cache misses. Hardware caching is fixed in the hardware or the operating system.
4. **Database Management Policies** : The database management system uses policies to handle different incoming calls to its databases. The database management system is tuned to perform good in common situations, but specific situations may occur in which it adds considerable overhead resulting in long waiting times.
5. **Deloitte INVision Caching** : The Deloitte INVision engine uses an internal cache to temporarily store results (dossiers) from the database management system. In some cases this scheme might not be suitable, resulting in long waiting times due to cache misses. But because functions are registered separately in the log database, if a cache miss occurs, the cache miss is not added to the completion time of the function in the log database, but registered as a separate function.
6. **Database Locking and Conflicts** : In a concurrent database environment, functions can read and modify the same data at the same time. To prevent faulty situations, functions might lock a database in order to keep the data consistent. These locking mechanisms could coincide with each other resulting in long waiting times, or failure of certain functions.

Database Locking and Conflicts (problem 6) is the only problem we can investigate in the context of this research. Due to the fact that we only have system data from the log database, in which minimal information considering the system state is logged (section 1.5), we can not conclude anything about the other problems. Problem 1 till 5 transcend the information contained in the log data, and we would need more data to conclude anything about them. Nevertheless, researching Database and Locking Conflicts contributes to the goal (section 1.2) which the research tries to fulfill.

1.5. Log data

Due to the fact that the information contained in the Deloitte INVision framework is considered confidentially, the possibilities to log the peculiarities of each function is limited. And due to the fact that there is only one log database, the information within the log database is chronological order, and contains the information about all functions of all solutions within the Deloitte INVision framework.

Although the amount of information contained in each log entry is limited, the following fields are available for analysis (see table 1.1).

STARTDATE	Information about the exact starting date of the function.
LOGFUNCALLID	The unique identifier of the function, this is a large integer.
LOGID	An unique identification belonging to the parent function of the current function. Sometimes a function starts other functions, the parent function id is then registered in the LogId.
PACKID	For some functions more information is logged concerning specific packs within a solution.
KSTID	The identification of the field, cell, row or document type from the Knowledge Specification Tool[6]. This value is only present for certain functions which operate on dossiers.
CALLNAME	The exact name of the function (or call) which was invoked by a user action.
SOLUTION	The name of the solution the function was operating on.
STARTTIME	The exact start time of the function, registered in milliseconds.
MICROSECONDS	The exact duration of the function, registered in microseconds, in the scope of the Deloitte INVision engine.

Table 1.1.: The information contained in each log entry

With the help of this information, the challenge is to find the solution for the problems which are discussed in the next chapter.

CHAPTER 2

RESEARCH PROBLEM

This chapter describes the main research problem of this thesis.

In the previous chapter an overview has been given about how the Deloitte IN-Vision framework is constructed, and which information is recorded in the log database. Also parts of the system in which failure or resource conflicts could originate are pointed out. The result of this research should not only communicate something bad is happening, but should also try to provide system operators with more detailed information concerning the nature of the problems. Considering the number of causes which could result in system failure or conflicts, and considering the minimal amount of available information it is impossible to discover the cause of all problems.

The first question one might ask, is what the *normal behavior* for the system is. What is the normal time an operation of type A takes, and how long does it take for a query of type B to return? Due to the ever changing state of the system these times are not fixed and cannot be modeled by a simple value. In order to solve this problem we need a technique which returns *normal behavior* but does this highly dynamic and time dependent. When the *normal behavior* is determined, we could find out which events take relative long times.

The next question is how the operating times of equal transactions behave in *concurrent versus sequential* operation. This information is highly usable for making predictions on how the system will react when the load is increased (section 1.2), more people start using the system, or larger data-operations are required. If we know how to determine normal behavior, then we might be able to determine normal behavior in sequential operation, and in concurrent operation. These both values can be compared, which tells us a little bit about the scalability of the system when adding more users: what might happen when most of the processing is done in concurrent operation.

Another important question concerning concurrency is ***if, and in what way, concurrent processes affect each others operation times***. This information can be used to implement efficient resource sharing and locking strategies for the system, and also an optimal processing order could be implemented using this information, which results in the best possible processing order. This is the most challenging problem. If we are able to somehow detect classes of concurrent op-

eration, we can find out which functions within concurrent operation result in a membership of a certain class. In figure 2.1 four example states[1] are depicted to show how some function calls (or events) could discriminate different classes.

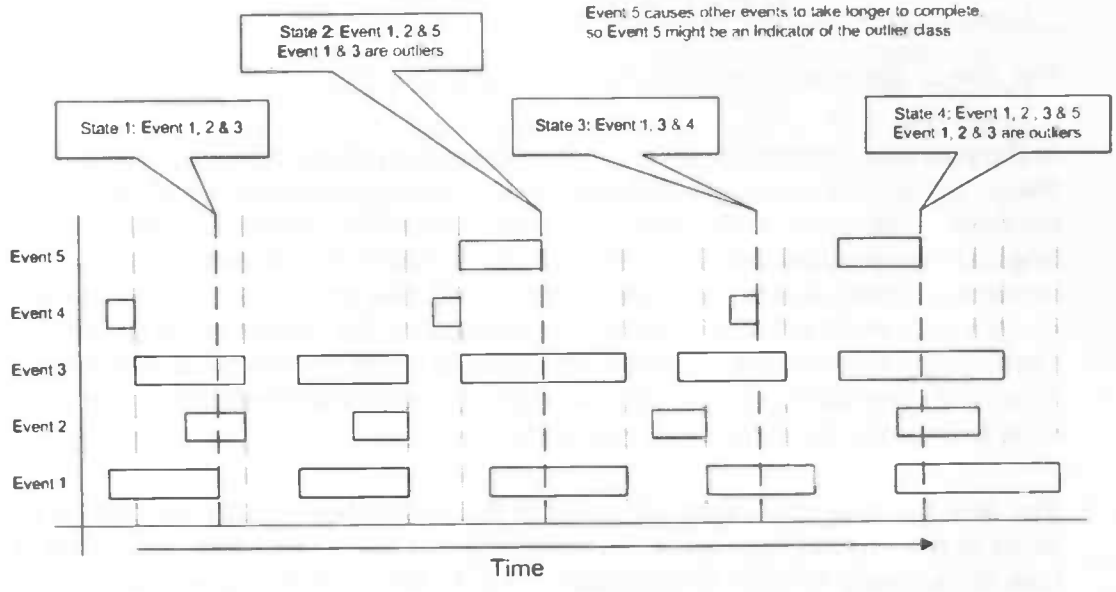

Figure 2.1.: Some function calls (or events) discriminate different classes

2.1. Generalizability

The solution to the research problem should be generic. It should be applicable for log based systems which are similar to the Deloitte INVision framework. We could set up problem preconditions which are mandatory in order to use the solution which is presented in this research paper.

[1]State : A measuring point at which we look at all the events which are currently in execution.

→ **Concurrency**	The current system has operations which are handled concurrently.
→ **Log-based**	The event information is logged, including the duration of each event.
→ **Resource Sharing**	The systems allows operations to interfere with each other, resources are shared.

Table 2.1.: General preconditions to use the solution presented in this research

2.2. Research Question

The previous problems can be stated as the main research question:

In an infinite series of events, how can we determine which events are outliers[2], and how can we predict combinations of concurrent processed events which cause exceptional processing times, using log-based event timing information.

In order to answer the research question, a couple of sub questions have been formulated.

2.2.1. Sub Questions

- For each transaction, how can determine which points are outliers using a dynamic measure to approach the *normal transaction time* in some time frame
- What is the relation between events processed concurrently and events processed sequential
- How can we find out if an anomalous event duration within a concurrent set of events can be due to resource conflicts or other problems.

[2]Outliers : Anomalous events

2.2.2. Approach

The search for answers to these research questions starts with literature research; the main research question suggests excavating information about outliers or anomalies. In order to predict combinations of concurrent processed events which cause exceptional times, we might look for other techniques which are able to recognize patterns, classify data or cluster data. These are well known techniques to perform knowledge extraction on data. The next chapter provides research information and previous work, and discusses the data mining techniques which might be feasible for a solution.

CHAPTER 3

PREVIOUS WORK

This chapter will expand our knowledge concerning different techniques and algorithms used for data mining, which have been developed by many other researchers.

Many researchers have investigated the field of data mining and knowledge discovery. Also system performance has been a subject of many scientific researchers. There is a wide varity of algorithms and solutions to a huge collection of data mining problems. Finding the right solution for this particular problem is not a trivial task.

3.1. Anomaly Detection

To solve the first subquestion (see section 2.2.1), we need a technique to determine points which lie far away from the other points. Points which are *"strange"*, the so called outliers or anomalous points.

Researchers all over the world have dug deep into this problem and as a result many anomaly detection algorithms and techniques have been developed.[Lan99] uses a technique called Temporal Sequence Learning to learn sequences from the data in order to detect which points are anomalous. Another technique used is State-Based Anomaly detection [Mic02], which uses states to describe how the system functions, unknown states result in anomalous points. [Min] introduces a Cache-based Anomaly detection algorithm which uses structures already created by the cache-protocol, where [Ste05] uses State Vector Machines to search for sets which are less concentrated in order to determine anomalies. [Yam00] presents an algorithm which uses Gaussian Mixture Models to determine anomalous points.

Problem with most these techniques is that they are less suitable for the problem which is discussed, although the Gaussian Mixture Models ([Yam00]) technique is promising for a solution to the problem of the first sub question.

Another technique which might be interesting, and which is researched intensively over the past decades is Data Clustering. Using data clustering it might be possible to also detect clusters of data and then somehow detect anomalous points. Data Clustering comes in many flavors.

3.2. Data Clustering

Clustering is one of the most widely used techniques to find discriminating classes in huge amounts of data. There are many types of clustering algorithms, but these can be divided in two main types of clustering:

- **PARTIAL CLUSTERING** [Jak80]: When partial clustering is used, all clusters are determined at once. So a single loop through the data is enough.
- **HIERARCHICAL CLUSTERING** : Hierarchical Clustering uses multiple passes through the data. Successive clusters are found using previously established clusters. Hierarchical clustering techniques can be bottom-up, or top-down.

Clustering is considered to be a form of unsupervised learning.
An interesting and promising technique is incremental clustering [Ash06], [Can93], which is based on the assumption that it is possible to consider data points one at a time and assign them to existing clusters. A new data item is assigned to a cluster without looking at the previously seen patterns. [Puz00] uses histogram clustering to optimize clusters, [Smy00] and [Jia06] use Probibalistic Clustering to determine the clusters. [Jia06] measure the deviation degree of a cluster in order to determine if the point is anomalous, which is a similar technique as [Yam00]. [Smy00] use Cross-validated likelihood to cross validate the clusters, thus creating a parameter free or unsupervised clustering algorithm. [Keo04] argues parameter free data mining is better. Also [Tas05] presents a solution for unsupervised detecting clusters in dynamic surroundings using an extension of the k-windows algorithm. [Sur05] introduces an unsupervised document clustering algorithm. In order to account for time data [Inn05] introduces seasonal clustering, and to deal with multi-dimensional data [Mon05] introduces an algorithm which is able to detect clusters in multiple dimensions. Using Coupled Clustering, [Mar03] tries to reveal equivalences (analogies) between sub-structures of distinct composite systems that are initially represented by unstructured data sets.

The two most well-know algorithms for clustering are K-Means [Mac67], which is an supervised clustering algorithm, and Expectation Maximization [Dem77] which is also a supervised clustering algorithm.

3.2.1. Expectation Maximization

The Expectation Maximization (EM) algorithm [Dem77] is an algorithm for finding maximum likelihood estimates of parameters in probabilistic models,

where the model depends on unobserved latent variables. EM alternates between performing an expectation (E) step, which computes an expectation of the likelihood by including the latent variables as if they were observed, and a maximization (M) step, which computes the maximum likelihood estimates of the parameters by maximizing the expected likelihood found on the E step. The parameters found on the M step are then used to begin another E step, and the process is repeated.

3.2.2. K-Means

The K-Means algorithm [Mac67] is an algorithm to cluster objects based on attributes into k partitions. It is a variant of the expectation-maximization algorithm in which the goal is to determine the k-means of data generated from Gaussian distributions. It assumes that the object attributes form a vector space. Then the algorithm tries to minimize total intra-cluster variance.

Although clustering sounds as a feasible solution for the first subquestion, [Lin03] states that **Clustering of Streaming Time Series is meaningless** (see [Lin03])
Clustering of streaming time series is completely meaningless. More concretely, clusters extracted from streaming time series are forced to obey a certain constraint that is pathologically unlikely to be satisfied by any dataset, and because of this, the clusters extracted by any clustering algorithm are essentially random.

Unfortunately we are dealing with sequential data which can be considered as streaming time series. So we can might say a clustering technique does not suit a feasible solution well.

3.3. Filtering

Filtering is a technique which also might be interesting in determining anomalous points.

Many Digital Filters have been developed, which use mathematical formulas to filter the input signal. If the input data is considered as the input signal, we might be able to filter and only return the outliers. [Sod03] presents a filter which uses time steps which can be adapted to suit the input data.
Although filtering might be a good solution for a lot of problems, due to the randomness of the data, it is very difficult to write a filter which filters outliers out effectively and correctly because filters are bound by mathematical models.

With the help of the three discussed techniques, we might be able to solve the first subquestion. The next step is to find a solution for the next two subquestions. When we have detected if a point is an outlier, we could try to detect patterns around this point.

3.4. Sequential Pattern Mining

A technique which could be applicable for resolving the third subquestion (section 2.2.1) is sequential pattern mining, which tries to recognize sequential patterns in the data. The problem here is how to determine those frequent sets which are interesting, while presenting a method which leads to a minimum computation and storage load requirement on the system. There has been a lot of research in sequential pattern mining, [Lak03] uses Dynamic Constrained Frequent Set Computation, which determines frequent sets in transactions. [Pei01] introduces a sequential pattern mining algorithm called PrefixSpan, which determines sequential patterns using an improved A priori technique. This technique is based on the fact that *any super-pattern of a non frequent pattern cannot be frequent*. [Che04] presents an incremental version of this algorithm called IncSpan.

Problem with pattern mining is that if we try to detect sequential patterns near anomalies, we do not have a clear definition of what *near* means. Although we could use incremental pattern mining, the solution would still require a huge amount of data storage the maintain pattern information. Considering we have more than 1000 different types of points, which could generate a huge number of patterns, pattern recognition might not fit very well for this problem.

3.5. Hidden Markov Models

Hidden Markov Models somewhat resemble to Sequential Pattern Mining. For an input stream a statistical model is created as a Markov Process [Rab89] with unknown parameters. The unknown parameters are estimated using the known parameters. Each state in the Markov Process has a model which determines the transitions which are possible. For Hidden Markov Models the same problems occur as in Sequential Pattern Mining, the number of states will explode resulting in enormous amounts of necessary storage.

3.6. Classification

Perhaps a better technique to solve the third research question might be classification.

As mentioned previously, classification is the procedure of placing items into different groups based on quantitative information on one or more characteristics inherent in the items. There are many different types of classification algorithms, the following algorithms might be interesting considering the research questions.

3.6.1. Support Vector Machines

Support Vector Machines (SVM) [Car01] try to classify the data with hyperplanes, which separate the data. Using optimalization techniques, the hyperplanes can be optimized with respect to the input data. SVM is an interesting technique to classify the data, although the representation of the hyperplanes and the optimalization process is somewhat complicated.

3.6.2. Learning Vector Quantization

Learning Vector Quantization [Bie06], [Vil06] is a technique which uses the input data in order to determine a so-called *prototype-vector*, which indicates the center of a certain classification class. Then, when a new data point is read, the *prototype-vector* is moved closer to the new data point if the classification class of the data point is the the same, otherwise the *prototype-vector* is moved away. Learning Vector Quantization can be implemented in a incremental way, which makes it easy to use.

3.6.3. Classification Expectation Maximization

[Cel92], [Sam05] present a modified version of the Expectation Maximization (EM) algorithm [Dem77] which is called the Classification EM (CEM) algorithm. The EM algorithm is modified by incorporating a classification step between the E-step and the M-step of the EM algorithm using a maximum a posteriori (MAP) principle. The original algorithm [Cel92] needs many scans through the data, the binned algorithm [Sam05] performs better.

These classification algorithms, except for CEM (section 3.6.3) all try to classify

the input data in such a way that the best classification is found. When anomalous points are found, all classification algorithms could perform a classification. Some might perform better considering the input data, but in the general case, the choice of the algorithm does not matter.

3.7. Preferred Algorithms

In the previous sections we have described many techniques which are promising for a solution to our main research question and the sub questions. To make the solution as general as possible (section 2.1), we chose to create a feasible solution by dividing the main problems in to three subproblems. This way similar problems can also be tackled by adjusting the proposed techniques, or by adding or removing algorithms. Considering the nature of the two subproblems we first need a technique to determine if events are outliers, a technique to detect an combination, and a technique to determine whether a combination of concurrent points results in outliers.

For the first problem, we explained 3 techniques (anomaly detection, data clustering and filtering) which could be applied. Because we are dealing in this case with time series data clustering drops out, and because we cannot model the data by a mathematical function also filtering drops out. In section 3.1 we introduced 5 anomaly detection algorithms : [Lan99],[Mic02], [Min], [Yam00] and [Ste05]. [Lan99] and [Mic02] do not suffice because the number of sequences can be infinite, [Min] also does not suffice while we do not have a cache based system. Because the memory requirements of [Ste05] might become very large, we will use the algorithm introduced by [Yam00]. This algorithm provides us with the information which points are anomalous, at the expense of minimum performance and memory.

The second problem can be tackled using a straightforward solution by looking at the start- and end-times of the function calls.

For the third problem three types of algorithms are explained (Sequential Pattern Mining, Hidden Markov Models and Classification). The first 2 techniques might not be feasible because we have a large amount of patterns which can occur, and the results of Pattern Mining (or Hidden Markov Models) might be somewhat unclear. Classification on the other hand gives us a clear answer to questions like:
Which elements within a set of events make this set belong to a certain class of sets

We explained two Classification algorithm, and because of its simplicity we will use Learning Vector Quantization. The use of LVQ alone does not provide us with enough information for a solution to the main question so we will use Relevance Learning Vector Quantization.

Now that we have chosen techniques we are going to use, these techniques first will be explained in the following two chapters.

CHAPTER 4

MODELLING GAUSSIANS

The previous chapter gave an overview of the relevant and most used techniques to perform data mining and knowledge discovery on the input data. This chapter will deal with the choices and algorithms which have been used to implement a solution for the first and the second sub questions.

Given the input data and the system properties, we will split the main problem into two subproblems.

- Identify which function calls within the input-stream are outliers.
- Identify if a function call occurs concurrent with other function calls.
- Classify the situations in which combinations of concurrent processed function calls occur. (as will be described in chapter 5).

As presented in the last chapter, there are three techniques which might be interesting to answer the first sub question. But for the first subproblem we use an outlier detection algorithm (see 3.7), for the second problem a straightforward solution, and the for the last problem a classification algorithm. Although some researchers have explored a similar way [Cel92], we can create a unique solution by combining fast and simple algorithms. This way we can keep the solution as general as possible. (see 2.1)

4.1. Anomaly Detection

If we look at the different algorithms described in the previous chapter, a good algorithm to use for the research problem would be the Online Outlier Detection algorithm [Yam00]. This algorithm uses Gaussian Mixture Models in combination with Expectation Maximization in order to search for outliers. The Online Outlier Detection algorithm is scalable because it only needs one scan through the data. During the scan the algorithm keeps track of a statistical model which is initial constructed and updated to fit the data.

Every time a new data point is inserted in the algorithm, the statistical model is updated. Then, out of the change in the statistical model a score is calculated. If this score is larger than a predefined threshold value, the algorithm concludes

the data point which was read is an outlier. To keep the statistical models time sensitive, aged data which have been read previously are gradually scaled out of the statistical model using a discounting parameter. This way only one scan through the data is needed. Considering [Lin03], the solution is valid, because we do not use a clustering algorithm to cluster the time series.

4.2. Correctness

A very important measure for the performance of the outlier detection algorithm, next to the amount of memory and computation time, is the amount of correct answers it returns. If the algorithm performs very fast, but returns many data points which actually are not outliers, the performance of the algorithm is poor. An optimal balance between full correctness and performance is required.

We can define some measures which indicate the correctness of the outlier detection algorithm:

$$\rho = \frac{\Upsilon}{Z - T} \times 100$$

where Υ is the number of false positives, and Z is the total number of data points, and T is the number of non-anomalies, ρ is a measure of correctness of the algorithm. Problem arises when a large number of data items is used, we need to validate the number of false positives by hand.

4.3. Distribution Approach

A big problem using the Online Outlier Detection algorithm [Yam00] is the definition of the statistical model. In the paper a Gaussian Mixture Model is introduced, but it could be the case that the input data does not fit well in a Gaussian Mixture Distribution. Other distributions might be interesting, such as a Poission distribution, or a chi-square distribution. During execution it is almost impossible to change the current statistical model to some other statistical model without loosing all knowledge of the statistical model which was constructed (using the previous data input points). So it is very important to pick the *correct* statistical model in advance, it the statistical model could be computed beforehand. Modeling the data with the *"bad"* statistical model does not necessarily produce bad results, but results are better if a model is used which suits the current data input distribution better.

4.3.1. Scaling

In order to improve the quality of the algorithm it is also possible to scale the input data before it is inserted in the algorithm. We could for example introduce a log-function to scale the input down in order to make the data input better fit the chosen distribution.

4.3.2. Number of Gaussians

Another problem for mixture models is that the number of gaussians (or any other distribution) which are needed to model the data is unknown beforehand. A mixture model looks as follows:

$$p(\mathbf{y}|\theta) = \sum_{i=1}^{k} c_i p(\mathbf{y}|\mu_i, \Lambda_i)$$

Where $p(\mathbf{y}|\mu_i, \Lambda_i)$ is the mixture model, c_i is a mixing-coefficient of the current distribution and k the number of distributions we use.

Cross-validation techniques have been developed to evaluate the log-likelihood of the chosen distribution [Mil03]. Using the KullbackLeibler divergence[1] afterwards, is a very computational expensive process, changing the models during runtime results in the same problem as before : all previous statistical models are removed and new statistical models need to be constructed again resulting in unnecessary true negatives in the output of the algorithm.

Within Deloitte INVision there are at the moment about 1000 different datatypes which could all have different distributions. In order to cope with this problem in the Deloitte INVision framework, and not to check all distributions by hand, a simple mechanism is used to quickly compute a rough estimation of the number of statistical models (gaussians) which are needed to model the distribution of each datatype (see chapter 4.5).

4.4. Moving Distribution

During the outlier detection with a mixture model, we maintain the state of the model using parameters μ, Λ and some other variables, where μ is the gaussian mean, and Λ is the covariance matrix. When a new data point is read, the current model needs to be changed to better fit the data. We want to create the best

[1]Kullback-Leibler divergence : a natural distance measure from a *true* probability distribution P to an arbitrary probability distribution Q

fit for the model of the distribution, but we require the algorithm to go only once over the data to update the model. See [Yam00] for detailed model updates. Adding a point which has a relative short duration results in moving the statistical model (μ) to the left or changes the variance (σ^2), adding a data point with a relative long duration results in moving the statistical model (μ) to the right (see figure 4.1), or changes the variance (σ^2). Adding a data point with *normal* duration does not change the statistical model.

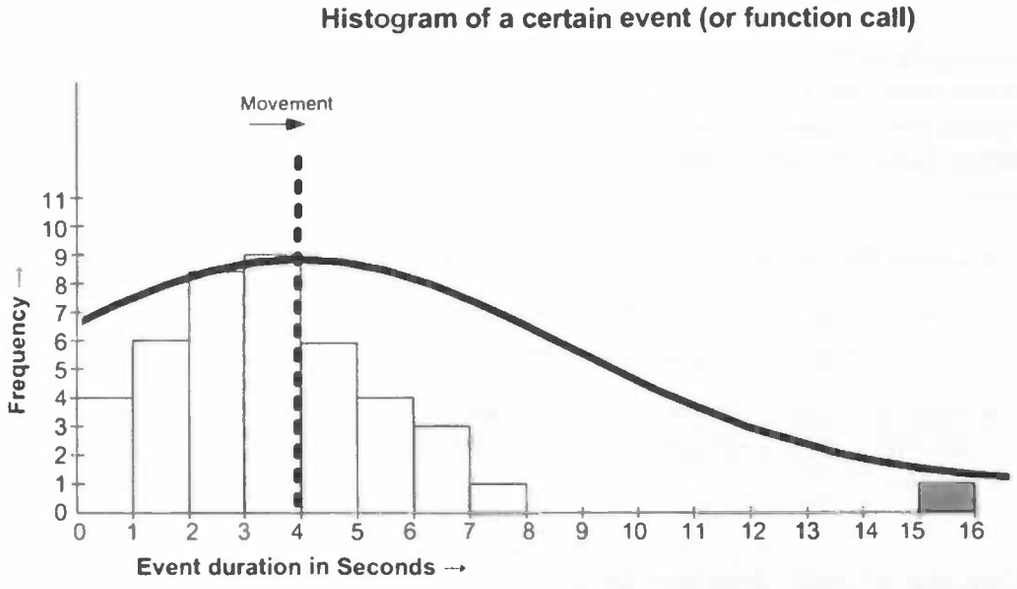

Figure 4.1.: Update the model by adding a point with relative long duration

The movement of the model can be interpreted as movement of the μ_i of the gaussians and extension of the variance σ_i^2 for all gaussians ($0 < i < k$) where k is the number of gaussians).

If the movement of the model is above or under some threshold, the outlier detection will return whether the current data point is a:

- *non-outlier* : a non outlier.
- *bad-outlier* : outlier which causes the mixture model (μ,σ^2) to move to the right. The data point has a long duration with respect to the *normal duration* of its class of data points. The system took a comparatively long time to process. This is the reason why we call it a **bad**-outlier.
- *good-outlier* : anomalous point which causes the mixture model (μ,σ^2) to move to the left. The data point has a short duration with respect to the

normal duration of its class of data points. The system took a comparatively short time to process. This is the reason why we call it a **good**-outlier.

These three types can be used for further analysis of the data point.

4.5. Binning

To roughly compute the number of Gaussians, we can use a binning technique on the data input histogram. We first need to find out a suitable *binsize*. A suitable *binsize* smooths out small humps in the histogram plot, and leaves big humps discoverable. When this is successfully detected the following could be done:

- Create bins of the sample distribution. If the smoothness does not suffice :
  - Average each bin with its left neighbor
  - Average each bin with it right neighbor
- Walk through the bins, if the derivative switches from increasing to descending, the number of Gaussians in the mixture model must be increased by one

When the anomaly detection algorithm is in place, and we can justify that it works correctly we can not only detect outliers in the input stream, but also at any point in time we could ask the system what the rate between outliers and non-outliers is.

The system load might be higher during concurrent operation, So we might expect that the number of outliers in concurrent operation is higher. We can now construct the following hypothesis:

4.6. Hypothesis 1

The percentage of encountered outliers during concurrent behavior is larger than the percentage of encountered outliers during sequential operation.

In order to test *Hypothesis 1* we need to setup a detection mechanism to detect whether a point is concurrent with other points.

4.7. Detection of Concurrency

Concurrency is easy detected. All we have to do is check at each measuring point[2] if previous measuring points overlap with the current point. Using the registered timing information from the log-database (see page 9) we could compute whether function calls are processed concurrently. If c_i and c_{i+1} are both successive function calls, c_i^{st} is the starting time of function call c_i, and c_i^{et} is the ending time, then if:

$$c_i^{st} + c_i^{et} \geq c_{i+1}^{st}$$

then c_i and c_{i+1} are concurrent function calls.

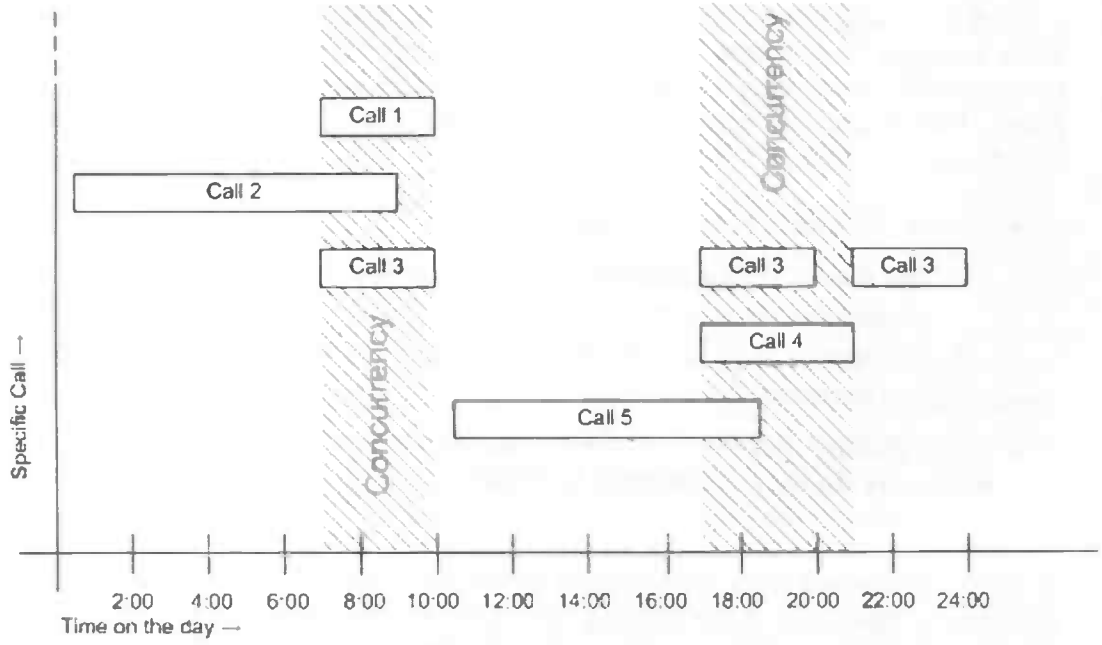

Figure 4.2.: Overlap of different function calls.

[2]Measuring point : each time a new input point is received

CHAPTER 5

CLASSIFICATION

This chapter will tell us which techniques and algorithms have been used for second part of the problem : classification of the results of the anomaly detection.

5.1. Definition of a concurrent state

Until now, we have setup an outlier detection scheme, which also registers if a point in time is concurrent. In order to answer the last part of the sub questions we need to *determine the combinations of points which often cause anomalies*, or equivalent, *determine the combinations of points in which often an anomaly can be found*. This equivalent question can be interpreted as a classification problem, which can be set up as follows:

- Divide the problem in two classes:
  1. The first class consists of sets of concurrent data input points, which contain no anomalous points.
  2. The second class consists of sets of concurrent data input points, which contain anomalous points.
- Try to classify these two sets: **what are the properties**, and more useful, **in what way do they differ from each other**.

A set of concurrent data input points represents a situation (or a state). This situation is measured at the starting time (c_i^{st}) of some function call i. The set σ can be interpreted as follows, where c_i^{st} is the starting time and c_i^{et} is the ending time of function call c_i:

$$\sigma = c_i^{st} \cap \sum_{i=1,j=1}^{k} ((c_j^{st} + c_j^{et} \geq c_i^{st}) \cap (c_j^{st} \geq c_i^{st}) \cap (j < i))$$

When we look at the classification algorithms presented in chapter 3.6 (see page 18) we can see that most classification algorithms will fit the problem. Because of its easy implementation, straightforward theory, and resemblance with the k-means algorithm the classification algorithm we will use is Learning Vector Quantization [Bie06].

5.2. LVQ in practice

In chapter 3 Learning Vector Quantization was explained shortly. In order to make it work for the Deloitte INVision problem we first take a closer look at the algorithm.
For each classification class Learning Vector Quantization creates a prototype vector which represents the center of the class. Figures 5.1, 5.2 and 5.3 show a clear picture of how Learning Vector Quantization works in practice. The result of applying Learning Vector Quantization multiple times is displayed in figure 5.4.

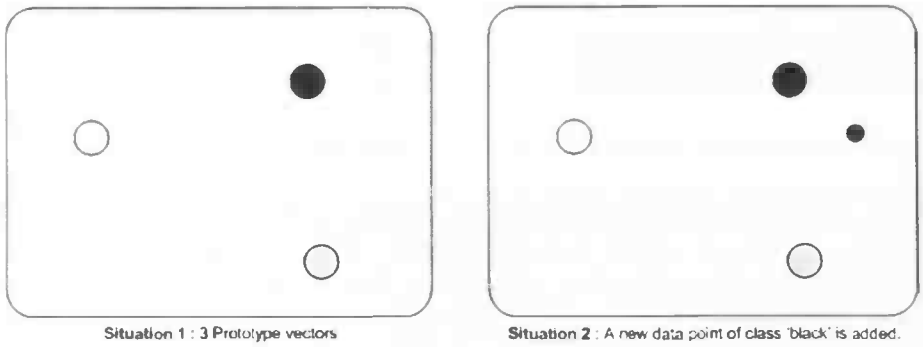

Figure 5.1.: Learning Vector Quantization : Prototype Vectors

In our situation each prototype vector represents a class which is *bad-outlier*, *good-outlier* or *non-outlier*. These classes relate to the three different outcomes the anomaly detection algorithm (chapter 4.1) returns.

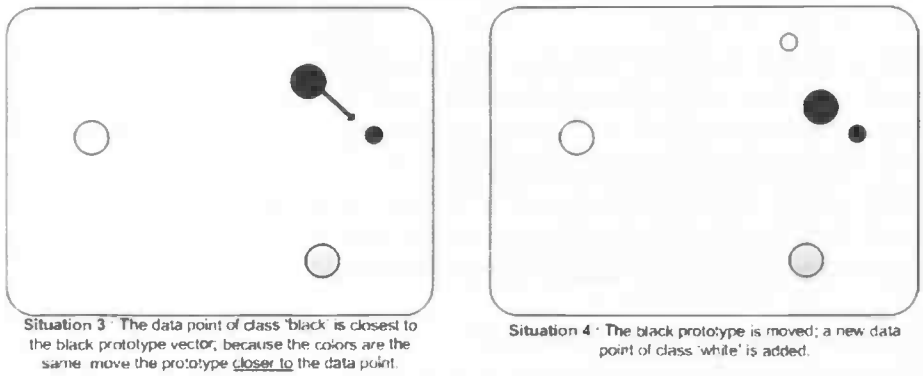

Figure 5.2.: Learning Vector Quantization : Movement of Prototype Vectors 1

During the learning vector quantization process, the prototype vectors are moved in n-dimensional space where n is the number of data types or unique function

calls (see 1.1). A good choice for n might be the number of *[solution,callname]* combinations.

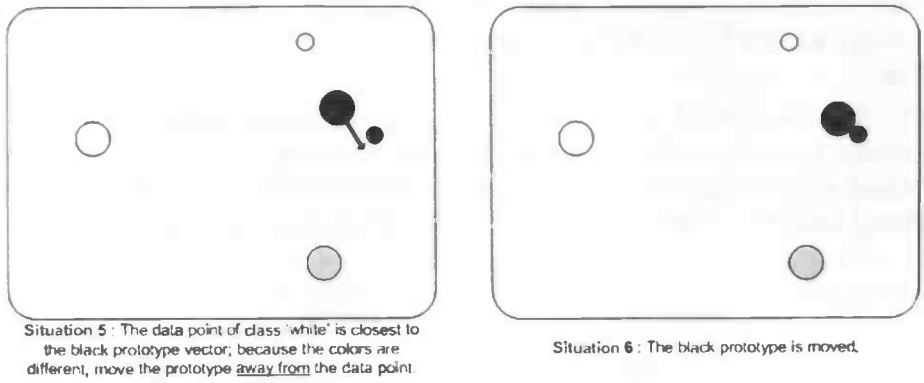

Figure 5.3.: Learning Vector Quantization : Movement of Prototype Vectors 2

The result of the continuous classification process is that we can extract the position of the prototype vector every moment. The position of the prototypes represents which unique function calls are important for the specific class. The difference between the bad-outlier prototype vector and the non-outlier prototype vector can be interpreted as the function calls which make a difference between these classes. Using this information, and the information from the anomaly detection algorithm, we are a good way in solving the third subquestion and the main research question.

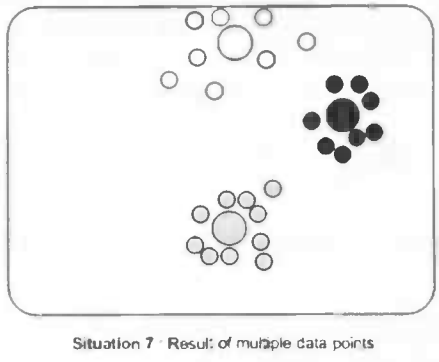

Figure 5.4.: Learning Vector Quantization : After a while, classes begin emerge

5.3. Relevance Learning Vector Quantization

The main research question states that *"how can we predict combinations of concurrent processed events"*, this means Learning Vector Quantization in its purest form (see 5.2) is not strong enough to solve this question. Learning Vector Quantization only tells us, given some situation $s_i = (c_1, c_2, ..., c_n)$ (where $(c_1, c_2, ..., c_n)$ are concurrent function calls), to which class s_i most likely belongs, however not presenting us with any information about which function calls in the list $c_1, c_2, ..., c_n$ are important indicators of the class membership.

To broaden our knowledge we use a modified form of Learning Vector Quantization which adds different weights to the dimensions of the prototypes vectors. This form of Learning Vector Quantization is called Relevance Learning Vector Quantization (RLVQ) [BHSvT], [HV02].

RLVQ is an enhancement of LVQ. It uses the same strategy for computation of the prototype vectors. Two modifications are applied to the original algorithm. The first modification is the distance measure, which we use to determine which prototype vector is the closest to the data input state (see 5.2). We make the distance also depending on the *weight* of the dimension (see section 5.4.3). The second modification is the update process of the relevance vectors.

5.3.1. Updates of the relevance vectors

The update of the relevance vector is executed every time a new data point is handled. The update procedure is showed in table 5.1).

So the relevance vector λ^k of the prototype vector δ^k which is closest to the data input point σ is first updated. If the class of the relevance vector is the same as the class of the data input point, then step (* 1 *) is performed, otherwise step (* 2 *) is performed. When all entries in the relevance vector are updated they need to be scaled again to suffice to the precondition : $((\sum_{i=0}^{n}(\lambda_i^k)) == 1)$, so we divide each entry in the relevance vector by the sum of all elements. Now the relevance vectors are nicely updated and apply different weights to the dimensions.

Because we do not know the exact number of function calls beforehand, we cannot exactly determine the length of the prototype and relevance vectors. When we initialize a relevance vector for each classification class, it could be the case that the relevance vectors assign different weights to the same dimension (or unique function call). This is caused by the scaling process. So when the num-

/** Update step **/
Forall relevance vectors $\lambda^1, .., \lambda^r$, prototype vectors $\delta^1, .., \delta^r$
and a state σ, where r is the number of classes, and $0 \leq j < r$:
if $(\min(\sqrt{\sum_{i=1}^{n}(\lambda_i^j(\delta_i^j - \sigma_i)^2)}) = \sqrt{(\lambda^k(\delta^k - \sigma)^2)})$ **then**
forall dimensions i :
$\lambda_i^k = \max(\lambda_i^k - \alpha|\delta_i^k - \sigma_i|, 0);$ **(* 1 *)**
else :
forall dimensions i :
$\lambda_i^k = \lambda_i^k + \alpha|\delta_i^k - \sigma_i|;$ **(* 2 *)**

/** Scaling step of λ^k **/
Forall dimensions i :
$\lambda_i^k = \lambda_i^k / \sum_{i=0}^{n} \lambda_i^k$ **(* 3 *)**

Table 5.1.: The relevance vector update procedure in RLVQ

ber of dimensions (or function calls) is not known beforehand we should use only one relevance vector. This does not influence the conclusions we make about the relevance learning vector quantization algorithm.

5.4. Used Data structures

In order to support the computation for classification, we need a data structure to support a n-dimensional space. We could interpret this n-dimensional space as a vector of length n, filled with floating point numbers (see figure 5.5).

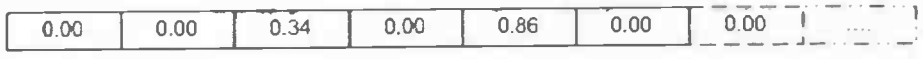

Figure 5.5.: A Sample prototype vector

During the Learning Vector Quantization process, the prototype vectors of the different classes are moved *toward to*, or *away from* the data input points. A data input point can be interpreted ass a set of function calls (chapter 4.7). A n-dimensional vector can be created of this set by starting with a vector containing all zeros. And then adding 1.00s to each vector-dimension supporting the unique *[solution,callname]*[1] combination for each function call from the set (see figure 5.6).

[1]See chapter 5.2

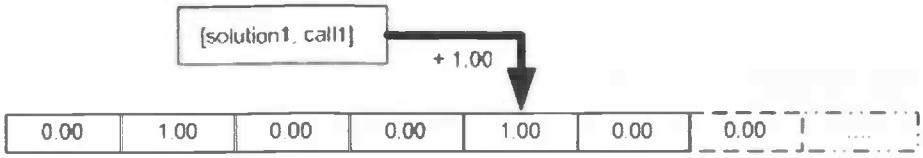

Figure 5.6.: A combination in a set of (solution,call) adds to the prototype vector

This way we create a vector which we can use in the classification process.

5.4.1. Initialization of the prototype vectors

The initialization of the Learning Vector Quantization algorithm requests that the prototype and relevance vectors are properly initialized. Setting the prototype vectors to all zeros does not work because the distance to the first data point is for all prototype vectors the same. Initializing the prototype vectors to first data input point of the same class as the prototype vector might be a better idea, but still we could initialize the prototypes to the same position. Therefore we need to verify after initialization, and during the execution of the algorithm, that prototype vectors do not completely overlap each other.

5.4.2. Setup of the dimensions

The relevance vectors for classification can be initialized by giving each dimension the same weight. This means every dimension has the same weight, which makes the distance measure (see 5.4.3) similar to the n-dimensional Euclidean Space. We achieve this by setting each entry in the relevance vector to $\frac{1}{vectorlength}$. The sum om all entries in the lambda vector ($\sum_{i=1}^{vectorlength}(\frac{1}{vectorlength})$) results to 1. This is efficient for scaling the vector during the update process.

5.4.3. Distance measure

The distance between a prototype vector and a data input point is measured using a modified version of the Euclidean Distance. The Euclidean Distance is a distance between two points $P = (p_1, p_2, ..., p_n)$ and $Q = (q_1, q_2, ..., q_n)$. This distance is defined in Euclidean Space as:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} \Rightarrow \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

For RLVQ we substitute this Euclidean Distance metric by:

$$\sqrt{\lambda_1(p_1 - q_1)^2 + \lambda_2(p_2 - q_2)^2 + ... + \lambda_n(p_n - q_n)^2} \Rightarrow \sqrt{\sum_{i=1}^{n} \lambda_i(p_i - q_i)^2}$$

Where λ_i is an entry from the λ-, or relevance vector.

This way the weight of each dimension is added to the distance measure. The prototype vector which has the smallest distance lies the closest to the data input vector. During the movement phase (section 5.4.4) this prototype vector is moved away, or moved toward the data input point, and its λ vector is adjusted (5.3.1).

5.4.4. Movement of the prototype vectors

Another conceptual problem arises when a prototype vector is moved away from a data input point due to class difference, which results that the prototype vector has a negative component for a certain dimension. The meaning of this negative component is as follows:

- When a component γ_i for some component i of the prototype vector γ is negative. It follows that for a set of concurrent events, there should be a *negative amount* of events of type i, in order to belong to the class of the prototype vector γ.

In the Deloitte INVision framework context it is not possible to have a negative amount of a certain function call in concurrent situations, but for the interpretation of the prototype vectors this makes sense. A negative value at a prototype vector position means that this dimension (or unique function call) should not be present in a concurrent situation, when the concurrent situation is of the same classification class as the prototype vector. For the relevance vectors this does not apply. Thus we have to prevent relevance vectors from becoming negative. So if a relevance vector becoming negative, we should set it to zero.

Now that we have discussed all techniques, let us continue with the implementation details of the prototype application.

CHAPTER 6

IMPLEMENTATION

This chapter will tell us the specific details of the implementation of the prototype which had been constructed to test the solution.

In order to implement a prototype to answer the main research question (see 2.2), we have to develop several algorithms and data structures which can handle vast amounts of data. In the previous chapters the process of knowledge discovery has been split-up in three steps:

- Detection if the current point was an outlier.
- Detection of concurrency.
- Classification of the situation in case concurrency is present.

Using these three steps we will explain the implementation of the prototype.

6.1. Used Data structures

To support the various computations which have to be carried out, we need for each step some data structures which help us to temporarily store the data.

6.1.1. Concurrency

Concurrency can be detected by comparing start and end-times of different incoming function calls. Due to the fact that function calls are always received sequential by the system, we can assume that :

If an function call e_i is received from the system at time τ_i, then the function call e_j, which is received at time τ_{i+1}, has finished later. When the data input points are retrieved directly from the Deloitte INVision engine this assumption is not valid, so other measures have to be taken to ensure this. In the prototype we receive the input data points from a csv-file (section 7.1), so we can ensure this assumption by sorting on **LogFunCallId** (section 1.5) ascending.

We use a circular buffer to *remember* information about a concurrent state. Using this circular buffer we can determine for each function call which function calls are concurrent. When we receive a new function call ϵ_i at time τ_i, we first

check the starting time of ϵ_i. To determine concurrency at measuring point τ_i we then need to go back in time and check for a certain number of function calls $(\epsilon_{i-n}, ..., \epsilon_{i-2}, \epsilon_{i-1})$ whether their end-times are larger or equivalent to the starting time of function call ϵ_i.

$$\forall(\epsilon_j^{et} > \epsilon_i^{st} \wedge j < i \wedge \epsilon_j^{st} < \epsilon_i^{st})$$

Where ϵ_i^{et} is the endtime of an function call i, and ϵ_i^{st} is the starttime.

6.1.2. Anomaly Detection

For each unique *[solution,callname]* combination[1] we can built a model which keeps track of the different gaussian mixture models as described in chapter 4.3. In order to save expensive memory we only maintain the following parameters for each model:

$nrcells$	The number of gaussians we want to use.
r	This is the discounting parameter for the current model.
α	A stability parameter for the SDEM algorithm.
$vector\_size$	The length of the vector containing a data point.
$dimension$	The number of gaussian we use to model the distribution.
c_i	The mixing coëfficient for the current model i.
γ_i	A temporarily parameter to store an interim result in the computation.
μ_i	The mean of the current gaussian mixtures.
$\overline{\mu}_i$	A temporarily parameter which is used during mean computation.
σ_i^2	The variance of the statistical model.
$\overline{\sigma}_i^2$	A temporarily parameter which is used during variance computation.

Table 6.1.: The parameters for the implementation.

A major problem with this scheme is that we need to maintain information about all the statistical models (*[solution,callname]* combinations) which have occurred during run-time. The models are stored in memory and accessible through a table of pointers.

6.1.3. Classification

The classification computation requires to store the state of the prototype vectors (chapter 5.4.1) and the relevance vectors. Every time we receive a new

[1] We use a *[solution,callname]* combination to generate a unique identifier for a statistical model. We could for example include *[solution,callname,packid]* in the measures, but this results in more models which have to be maintained.

state[2] we build an n-dimensional vector from it, and then perform the classification procedure. A challenge is to construct a data structure which is able to create dynamical n-dimensional vector which can be used during classification for both the prototype vectors, and the data input point vectors. This n-dimensional vector needs to be dynamic because the dimensions of the state (or unique *[solution,callname]* combinations) might change during execution. At the moment a simple static vector is used, but this could be improved.

6.2. Initialization

During initialization of the prototype, the prototype vectors are initialized, the circular buffer for the concurrency detection is initialized, and the vectors are created to maintain the models for the outlier detection.

6.3. Outlier Detection Implementation

The first big challenge was to implement the outlier detection algorithm. Because we are performing complex mathematical operations, the GNU Scientific Library[3] is included in the project.

The implementation of the outlier detection part is roughly following the example algorithm as proposed by [Yam00]. In their paper [Yam00] propose an algorithm called *Smartsifter* which consists of two parts, a categorical algorithm called SDLE[4] which assigns a input point to a certain class using a histogram, and a continuous algorithm called SDEM[5] which maintains the mixture models. Because for each data input point we know its class, we could skip the SDLE algorithm and directly use the SDEM algorithm to update the mixtures, because we only have one category for each model. The initialization of the SDEM algorithm is shown in table 6.2 on page 36.

Step 1 : /* Initialization */ (r, α, k given):
Set $\mu_i^{(0)}, c_i^{(0)}, \overline{\mu}_i^{(0)}, \Lambda_i^{(0)}, \overline{\Lambda}_i^{(0)} (i = 1, ..., k)$.
$t := 1$

Table 6.2.: The initialization of the SDEM Algorithm

[2] A state as discussed in section 6.1.1

[3] GNU Scientific Library : Free library for scientific and numerical work, including integration and equation solving.

[4] SDLE : Sequentially Discounting Laplace Estimation

[5] SDEM : Sequentially Discounting Expectation and Maximizing

The second part of the SDEM algorithm is shown in table 6.3 on page 37.

Step 2 : /* Parameter Updating */ :
while $t \leq T$ (T : sampling size)
Read y_t
for $i = 1, 2, ..., k$

$$\gamma_i^{(t)} := (1 - \alpha r) \frac{c_i^{(t-1)} p(y_t | \mu_i^{(t-1)}, \Lambda_i^{(t-1)})}{\sum_{i=1}^{k} c_i^{(t-1)} p(y_t | \mu_i^{(t-1)}, \Lambda_i^{(t-1)})} + \frac{\alpha r}{k}$$

$$c_i^{(t)} := (1 - r) c_i^{(t-1)} + r\gamma_i^{(t)}$$

$$\overline{\mu}_i^{(t)} := (1 - r)\overline{\mu}_i^{(t-1)} + r\gamma_i^{(t)} \cdot y_t$$

$$\mu_i^{(t)} := \overline{\mu}_i^{(t)} / c_i^{(t)}$$

$$\overline{\Lambda}_i^{(t)} := (1 - r)\overline{\Lambda}_i^{(t-1)} + r\gamma_i^{(t)} \cdot y_t y_t^T$$

$$\Lambda_i^{(t)} := \overline{\Lambda}_i^{(t)} / c_i^{(t)} - \mu_i^{(t)} (\mu_i^{(t)})^T$$

Table 6.3.: The second step of the SDEM Algorithm

The parameters are the same as in the initialization (6.1.2). This algorithm is implemented straightforward in the C language[6].

6.4. Classification

Our classification uses three classes which denote *good-outliers*, *bad-outliers* and *non-outliers*. The classification procedure is only executed when a concurrent situation occurs, otherwise the data input state[7] contains $(n - 1)$ zeros and (1) one. The distance between the data input vector and the prototype vectors is measured as in section 5.4.3, the prototype vector which has the smallest distance with respect to the data input point is moved, and the relevance vector is updated and scaled. Given data input point δ of class τ and prototype vectors of types $(X_0, X_1, ..., X_n)$:

- If

$$min(\sqrt{\sum_{i=1}^{n} (X_i - \delta)^2}) == (X_j - \delta)$$

  and $X_j^\tau == \delta^\tau$ where τ is the type of the class, then move X_j closer to δ.

- Otherwise move X_j away from δ.

[6]The C programming language is a standardized programming language developed in the early 1970s.
[7]State : a similar data structure as a prototype vector as described in 5.4.1

This way prototype vectors are moved in the n-dimensional space. To make sure that the relevance vectors do not reach values below zero, we have to set them to zero if they are less then zero (5.4.4). In the case above the relevance vector is updated and scaled again as in section 5.3.1.

6.5. Overall Implementation Impression

We can conclude that the implementation of the prototype has various data structures to efficiently support both algorithms. Every time a new data input point is read, the concurrency detection processes the point, then the outlier detection mechanism checks whether the point was a outlier, and the classification algorithm uses the point in combination with other points to update the prototype vectors and the relevance vectors.

The next chapter will give us information about the results this prototype produced using a large input set of real production data from the Deloitte INVision framework. For each separate algorithm the results will be explained and the overall performance will be measured.

CHAPTER 7

RESULTS

This chapter will present the result of testing the prototype application on real data from the Deloitte INVision framework.

Now that we have created a prototype we need to test it. The best test data is real log data from the Deloitte INVision engine. A large database of approximately 40 Gigabytes was used as a test set for the tests.

Correctness (chapter 4.2) is an item we cannot test while handling large quantities of input data. This is due to the fact that we simply cannot cross-validate all the results by hand. Also correctness is a relative conception, given the input data, it is unclear what an outlier is, and what not. The only tests for correctness can be done by visual inspection of the results.

Performance is an test item we can test really good. By measuring the duration of some operation we can compute the average time the prototype needs for a data input point to compute. Also the consumption of system resources can be monitored using the task manager.

7.1. Test sets

In order to test the prototype application we created 2 test sets, which consisted of real data from the Deloitte INVision framework.

- The first test consisted of the log-database which has a size of approximately 6 Gigabytes.
- The second test consisted of the log-database which has a size of approximately 40 Gigabytes.

Each database has to be prepared beforehand and needs to be converted to a *comma separated file*-format. This because the prototype reads directly from file to minimize the performance degradation through read operations.

In debug mode the prototype produces some files which contain information about the prototype vectors, the relevance vectors, and the found outliers. In performance tests this debug has been turned off to improve performance.

7.2. Performance Measures

Performance measures were measured using a Intel Pentium M 1.6 Ghz processor, with 1 Gigabyte of RAM. The results of the performance test are shown in table 7.1.

6 Gigabyte Dataset (12.619.729 **function calls)**	**40 Gigabyte Dataset** (50.860.254 **function calls)**
first run: Computation time : 2519 seconds Memory consumption : 4.5 Megabytes Time per call : $\frac{2519}{12619729} = 199$ μseconds	**first run:** Computation time : 59921 seconds Memory consumption : 5.2 Megabytes Time per call : $\frac{59921}{50860254} = 1178$ μseconds
second run: Computation time : 2462 seconds Memory consumption : 4.5 Megabytes Time per call : $\frac{2462}{12619729} = 195$ μseconds	**second run:** Computation time : 59102 seconds Memory consumption : 5.2 Megabytes Time per call : $\frac{59102}{50860254} = 1162$ μseconds
Average: Computation time : 2491 seconds Memory consumption : 4.5 Megabytes Time per call : $\frac{2491}{12619729} = 197$ μseconds	**Average:** Computation time : 59512 seconds Memory consumption : 5.2 Megabytes Time per call : $\frac{59512}{50860254} = 1170$ μseconds

Table 7.1.: Performance on the different data sets

From table 7.1 we can conclude the memory consumption per call does significantly change when a larger dataset is taken. Thorough inspections of the behavior of the prototype showed us that this problem was due to a hashing-mechanism, which was used to uniquely identify the *(solution, callname)* combinations. When this hashing is removed, the prototype becomes significantly faster. When the current prototype is built into the Deloitte INVision engine, in its current state, it will approximately take 1200μseconds of computation time to perform outlier detection and classification. Within production environment the capacity of the hardware is much larger than the 1.6 GHz test system, so computation time will be less than 1200μseconds in production. When the prototype is improved by a better hashing algorithm, this can be improved significantly.

7.3. Outlier Detection

The result of the outlier detection consists of a long list of items. Each item that has been scanned by the prototype is marked with a class (*non outlier,good*

outlier or *bad outlier*) (section 4.4). This list has the same size as the number of data points which had been put into the prototype application.

CallName	MicroSeconds	Class
OnFinishEditing	182	non outlier / good outlier
LoadDossier	3843	outlier
Background:AConsolRev	3887	outlier
Background:AConsolRev	3785	outlier
GetDossierIndexXML	269	outlier
Background:AConsol	25348	outlier
OnFinishEditing	217	outlier
GetAncestorIDs	4289	non outlier / good outlier
GetDossierIndexXML	311	non outlier / good outlier
GetPath	3773	non outlier / good outlier
GetDocumentXML	4158	non outlier / good outlier
ExecSelectSP_Select104_GetAttachments	5769	non outlier / good outlier
GetAncestorIDs	7513	outlier
GetDossierIndexXML	270	outlier
GetPath	4635	outlier
GetDocumentXML	1561	outlier
GetDossierIndexXML	50004	outlier
LoadDossier	680433	outlier
GetIndexXML	133887	non outlier / good outlier
GetIndexXML	39649	outlier
EGetDossierIndexXML	15999	outlier

Figure 7.1.: A sample of the results from the outlier detection algorithm

When we plot this list, and then filter as a specific (*solution*, *callname*)-combination, we get nice results (figure 7.2 42), to create this plot we used the following parameters for the outlier detection algorithm (table 7.2 on page 41).

Parameter :	**Value :**
thightness	0.001
threshold	2.8
rlvqlearningrate	0.001
rlvqlearningupdaterate	0.001

Table 7.2.: Outlier detection parameters

When we look at figure 7.3 (page 42), we can clearly see all the large points are filtered out, and marked by the outlier detection algorithm. In the beginning of the graph, we see a lot of outliers. These are caused by the configuration of the algorithm, the outlier detection algorithm has to calibrate. During this

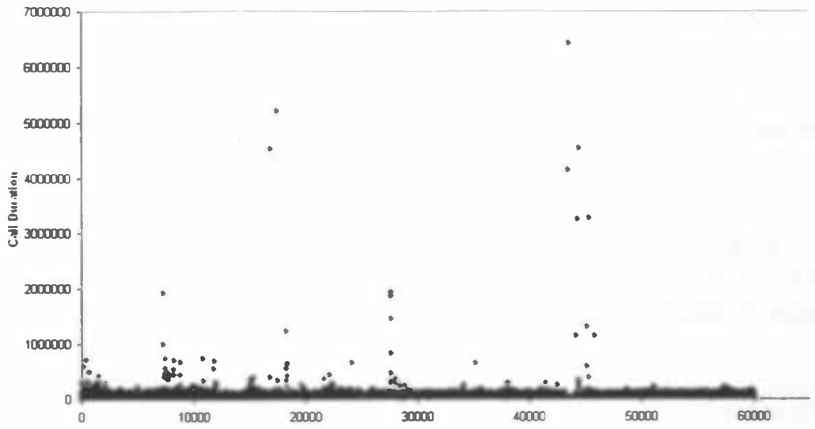

Figure 7.2.: Durations of the call GetIndexXML on a solution

calibration phase points that lie a little bit away can change the model drastically, which results in a high number of false outliers. When the algorithm has calibrated, the outlier detection is better and high values are detected by the algorithm.

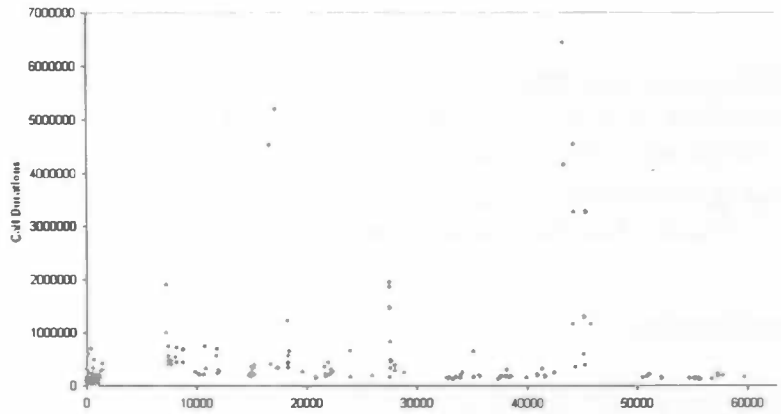

Figure 7.3.: The found outliers in the (*solution,callname*)-combination from figure 7.2

It is also good to see in figure 7.3 (page 42) that the threshold value of outlier versus non outlier values varies over time. The models learn the distribution over time, so detection becomes better and better.

7.4. Concurrency Detection

Concurrency is the second point we have tested. In a concurrent situation multiple points are running at the same time. If debug mode[1] is activated, a file is created which shows us all the concurrent events for each event.

For the two large test sets we also have concurrency statistics available. For each test set the number of concurrent items, and the number of non-concurrent function calls is printed to a file. The results of both test sets are shown in table 7.3.

6 Gigabyte Dataset	**40 Gigabyte Dataset**
Nr. Concurrent Items : 3512960	Nr. Concurrent Items : 19163093
Nr. Non Concurrent Items : 9106769	Nr. Non Concurrent Items : 31697161
Nr. Concurrent Outliers : 203176	Nr. Concurrent Outliers : 837895
Nr. Non Concurrent Outliers : 238619	Nr. Non Concurrent Outliers : 1129598
Percentage Concurrent Outliers : 5,78	Percentage Concurrent Outliers : 4,37
Percentage Non Concurrent Outliers : 2,62	Percentage Non Concurrent Outliers : 3,56

Table 7.3.: Concurrency statistics on both test sets

From table 7.3 we can see that the percentage of concurrent outliers is larger than the percentage of non concurrent outliers. This means hypothesis 1 (see section 4.6) can be answered positively.

The percentage of encountered outliers during concurrent behavior is larger than the percentage of encountered outliers during sequential behavior.

7.5. Classification

The last part of the prototype consists of classifying concurrent situations, and creating prototype and relevance vectors from it. The standard way to do this is by creating a training set, and a validation set. The validation set is left away, and we only concentrate on the training set. While scanning the data input points, we gradually construct the prototype vectors for all classes (*non outlier, good outlier* and *bad outlier*). At the end of the run, the prototype then prints all components of these prototype vectors to a file. The size of the prototype vectors is set to a maximum of 1000 dimensions.

[1] The prototype is in debug mode if debug data is printed to different files

non	bad	good	relevance
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0,001291
-4,2E-05	1	1	0,003092
0	0	0	0,002352
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0,002692
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0,003	0	0	0

Figure 7.4.: A sample of the result of the classification procedure, the gray-colored row has high relevance

As we can see in figure 7.4 (page 44), the prototype vectors and the relevance vectors have different values.

7.5.1. Values of the prototype vectors

Every value in the prototype vector gives us information about a dimension[2]. If a prototype vector value is 0, this means that this dimension rarely applies for this class. If the value of the prototype vector at position i is larger than 0, this means the $(i)^{th}$ (*solution,callname*)- combination often occurs one or more times in a concurrent situation which belongs to the class of the prototype.

Example:

Suggest we have a prototype vector of class A, and a prototype vector of class B. Each prototype vector has 3 dimensions. Suggest we have one Solution : S1, and we have

[2]A dimension is a unique (callname,solution)- combination

three function calls : C1, C2, C3. Each dimension in the prototype vector stands for a unique (solution,callname) combination: dimension 1 = (S1,C1), dimension 2 = (S1,C2) and dimension 3 = (S1,C3). We look at the prototype vector of class A after the program has executed for a while. If the value in the first and the third dimension of the prototype vector A are zero, this means class A contains zero function calls C1 and C3 of solution S1. If the value in the second dimension of prototype vector A is two, this means that class A usually contains two function calls from solution S1, so a concurrent situation then looks like this: (" ",C2,C2," "). When we have looked at all dimensions, we can see what function calls are usually in what class.

7.5.2. Values of the relevance vectors

Similar to the prototype vectors, every value within the relevance vectors give us information about a different dimension. The difference between the prototype vectors and the relevance vectors is that the relevance vectors tell us which *solution,callname*-combination (or dimensions) are important for membership of a certain class. If the value of a relevance vector is high according to the rest of the relevance vector values, this means this dimension is an important **indicator** for a certain class.

In our situation this means if a relevance vector contains high values, the (*solution,callname*)- combination leads in concurrent situation often with / or without other function calls to a certain class.

THE ABOVE STATEMENTS ONLY SUGGEST THAT IF A RELEVANCE VECTOR HAS A HIGH VALUE AT POSITION [SOLUTION,CALLNAME], THEN THIS [SOLUTION,CALLNAME]-COMBINATION HAS A HIGH CONTRIBUTION TO A CERTAIN CLASS MEMBERSHIP, BUT IT DOES NOT SAY ANYTHING ABOUT WHICH EXACT COMBINATIONS OFTEN RESULT IN AN OUTLIER.

At the end of the test, the relevance vectors where printed to a file. We will show the top 2 (*solution, callname*)-combinations, which have the highest values in the relevance vectors, in table 7.4.

	Function Call	**Relevance**
6 GB Dataset	1. Background:AConsol	0,298289
	2. Background:RCpyAnswer	0,119492
40 GB Dataset	1. IsFunctionAllowed	0,251374
	2. Background:PackUpgrade	0,104075

Table 7.4.: Highest dimensions in the Relevance vectors

There are many problems, these can be figured out by closely identifying the

relevance vectors.

7.6. Prototype Application Results

Using the results we successfully confirmed hypothesis 1, showing that the percentage of outliers during concurrent execution is higher than in sequential execution. The results also showed us that it is very hard to discover exactly which function calls or (solution,callname)- combinations are responsible for outliers. This is due to the fact that the definition is somewhat unclear, and the number of possibilities is enormous. The next chapter will continue on this subject.

CHAPTER 8

DISCUSSION

This chapter will reflect the results from the previous chapter to the original research questions, do the results satisfy enough, and where are sections which should be researched further.

So did the results really satisfy the research questions (section 2.2), and is the research successful considering the research goals. In order to answer these questions we first look at the three subgoals which were introduced.

8.1. Subquestions

- **Q1** : For each transaction, how can determine which points are outliers using a dynamic measure to approach the *normal transaction time* in some time frame
- **Q2** : What is the relation between events processed concurrently and events processed sequential
- **Q3** : How can we find out if an anomalous event duration within a concurrent set of events can be due to resource conflicts or other problems.

After the introduction of these research subquestions we introduced data mining and knowledge techniques. We selected two techniques to perform the appropriate analysis, and set these techniques up in a prototype application. In order to make the solution as general as possible we choose not to use an all-in-one solution (where all separate techniques are interweaved with each other), so that researchers can improve or update the techniques separately. Before the research really started, we chopped the main research question into three sub-problems. The first problem had to do with outlier detection or outlier detection, the second with concurrency and the third problem with classification. In order to discover concurrency we added a concurrency detection mechanism in the prototype.

When we look at the results of the prototype, and compare these to the research subquestions we get the following result.

- **Q1** : Subquestion one has been fulfilled using the outlier detection scheme, the usage of Gaussian mixture models introduces a measure for the *normal*

transaction time by means of a statistical model. The graduate discounting of time for this model introduces its time-sensitiveness.

- Q2 : The second subquestion can be answered using the results of the concurrency detection mechanism. The number of outliers and non-outliers is registered. A simple computation could show us the relation between function calls processed concurrently and function calls processed sequential. Also hypothesis 1 (section 4.6) is satisfied using this information.
- Q3 : The last subquestion is mostly answered by the classification procedure which is introduced in section 5.3. The classification procedure finds out whether different function calls are **indicators** for a set of concurrent function calls including an outlier. Considering the many causes a problem can be due to (section 1.4), it is almost impossible to exactly determine the cause of the problem. Experiments (section 7.5) show us that we can identify problems which are due to resource conflicts.

When we look at the mapping between the research subquestions, and the results of the prototype we can say that except for the classification part, the research subquestions can be answered sufficiently. The techniques which could be used to answer subquestion Q1 and Q2 have have been formulated. The last research subquestion is partly satisfied. Using the theory and the prototype application we can show that we can detect which events within a concurrent situation have high (or low) contribution to the fact of a function call being an outlier, but we did not succeed in finding the exact combinations, while the number of combinations remains large.

8.2. Further Research

The last section we evaluated the results of the prototype in accordance with the research subquestions. We succeeded in finding a solution for the first two research subquestions, and partly answered the third research subquestion. The problem of this third research subquestion is that the number of combinations can be huge. *Does it make sense to return a huge number of combinations which represent outlier situations?* This is one of the questions which still have to be answered.

There are a few more areas within this research which still could require further research in order to improve results. Starting with the definition of outliers.

8.2.1. Definition of Outliers

Definition of outliers; the most important definition of this research. Given a list of occurrences of a function call, and two classes : (outlier, non-outlier). Assign a class to a new occurrence of the function call. The difficulty is that we can define mathematical procedures, statistical models or other techniques, but still we could classify a function call in the wrong class. Due to the fact the input data is multivalued it is impossible to directly create a strategy which tells us about the class of the event. We used a Gaussian Mixture Model to determine whether a data input point is an outlier, but it might be interesting to research other techniques and their results. Still, the human perception of data input points is very important.

8.2.2. Correct Model Distribution

Another issue in the outlier detection algorithm which should be researched further is the choice of distribution. We chose for a Gaussian distribution, but another distribution might fit better or worse. The Kullback- Leibler distance (see section 4.3) is a good measure to validate this, and an algorithmic approach using this distance measure might suit the prototype well. The use of different distribution has effect on the prototype code, so this should be updated. Fitting the best distribution for the data results in the best outlier detection, at the expense of performance loss due to the search for the best distribution.

8.2.3. Number of Gaussians

The number of Gaussians we use in the prototype is set to one, but in order to better suit the distribution we might need more Gaussians, or in the case of an other distribution (section 8.2.2) more models. Further research might concentrate on a way to incorporate the *choice of the number of models* in the outlier detection algorithm. Of course the performance should not decrease significantly due to this added computation.

8.2.4. Correct Classification Algorithm

Finally, the most challenging part of this research which should be researched further is the classification. It might be possible to somehow change the distance measure (section 5.4.3) and get instead of the relevance vector a relevance matrix. With regression methods, it might be possible to detect which exact combinations of dimensions are important.

In section 8.2.1,8.2.2,8.2.3 and 8.2.4 the most evident open issues in this research are discussed, but there might be other interesting research directions which elucidate new understandings in this matter.

CHAPTER 9

CONCLUSION

This chapter will formulate the final conclusion, both to the main research question, as the subquestions.

This research starts with the introduction of the research environment. The Deloitte INVision framework, which serves as an source for the research problem, is able to handle multiple function calls concurrently, and uses a log database to store information about every function call. Because the information contained in the function calls is confident, the log records only contain minimal information. The goal of this research is to structurally extract new knowledge from this log database, in order to do prediction on performance issues. Issues like what operation times significantly differ from normal operation times. This goal is recorded in the main research question.

In an infinite series of events, how can we determine which events are outliers, and how can we predict combinations of concurrent processed events which cause exceptional processing times, using log-based event timing information.

In order to solve this research question three subquestions were defined to structure the search for an overall solution.

- For each transaction, how can determine which points are outliers using a dynamic measure to approach the *normal transaction time* in some time frame
- What is the relation between events processed concurrently and events processed sequential
- How can we find out if an anomalous event duration within a concurrent set of events can be due to resource conflicts or other problems.

We can also formulate the hypothesis that outliers occur significantly more often in concurrent system operation.

The next part of the research consist of literature exploration. Different techniques are mentioned and reviewed according to the Deloitte INVision framework. Not all techniques are applicable, and we chose those techniques which seemed to best fit the framework and are as generic as possible. Following

the research subquestions, a generic solution is constructed which is split up in three parts : Outlier Detection, Concurrency Detection and Classification.

For outlier detection we use the Smartsifter algorithm, an algorithm which uses Gaussian Mixture Models to determine whether a data point is an outlier, or not. There are many classification algorithms, but we use Relevance Learning Vector Quantization. This algorithm is easy to implement, and produces good results.

The implementation of these algorithms is carried out by creating a prototype application, which was tested on two large real world datasets. During the tests the prototype performed good with regard to memory and processor usage. The results of the outlier detection and concurrency detection scale very well, and the hypothesis is confirmed using the results. Classification performs good, while the value of the results is somewhat less relevant. More research has to be carried out to clarify these results even further. So the first two research subquestions are satisfied, and the third partly.

Considering the research results we can say the main research question can be satisfied partly. Using the proposed solution we are able to determine which function calls are outliers, and we are partly able to determine whether a combination of processed function calls (or a certain function call in concurrent situation) which causes an exception (which is an indicator for the exception-class) can be predicted.

For Deloitte, this research gives more insight in outlier detection. At the moment, this is done by a static value; if a function call takes longer than three seconds to complete, it is reported as an error. This could be improved using the outlier detection algorithm from this thesis. The results of the concurrency detection and the classification might give the system administrators more insight into the system.

Overall we can say this research has mostly solved the research questions, and fulfilled the goals of the research.

BIBLIOGRAPHY

[Ash06] Asharaf, S. & Narasimha Murty, M. & Shevade, S.K. Rough set based incremental clustering of interval data. *Pattern Recognition Letters*, 27(6):515 – 519, April 2006.

[BHSvT] T. Bojer, B. Hammer, D. Schunk, and K. von Toschanowitz. Relevance determination in learning vector quantization. In *ESANN'01*.

[Bie06] Biehl, M. & Ghosh, A. & Hammer, B. Learning vector quantization: The dynamics of winner-takes-all algorithms. *Journal of Neurocomputing*, 69(7 - 9):660 – 670, March 2006.

[Can93] Can, F. Incremental information clustering for dynamic processing. *ACM Transactions on Information Systems*, 11(2):143 – 164, April 1993.

[Car01] Caragea, D. & Cook, D. & Honavar, V.G. Gaining insights into support vector machine pattern classifiers using projection-based tour methods. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 251 – 256, 2001.

[Cel92] Celeux, G. & Govaert, G. A classification em algorithm for clustering and two stochastic versions. *Journal of Computational Statistics & Data Analysis*, 14(3):315 – 332, October 1992.

[Che04] Cheng, H. & Yan, X. & Han, J. Incspan: incremental mining of sequential patterns in large database. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 527 – 532, 2004.

[Dem77] Dempster, A.P. & Laird, N.M. & Rubin, D.B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1 – 38, 1977.

[HV02] B. Hammer and T. Villmann. Batch-rlvq, 2002.

[Inn05] Inniss, T.R. Seasonal clustering technique for time series data. *European Journal of Operational Research*, August 2005.

[Jak80] Jakobsson, M. Reducing blok accesses in inverted files by partial clustering. *Journal of Information Systems*, 5(1):1 – 5, 1980.

[Jia06] Jiang, S. & Song, X. & Wang, H. & Han, J. & Li, Q. A clustering-based method for unsupervised intrusion detections. *Pattern Recognition Letters*, 27(7):802 – 810, May 2006.

[Keo04] Keogh, E. & Lonardi, S. & Ratanamahatana, C.A. Towards parameter-free data mining. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206 – 215, 2004.

[Lak03] Lakshmanan, L.V.S. & Leung, C.K. & Ng, R.T. Efficient dynamic mining of constrained frequent sets. *ACM Transactions on Database Systems*, 24(4), December 2003.

[Lan99] Lane, T. & Brodley, C.E. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295 – 331, August 1999.

[Lin03] Lin, J. & Keogh, E. & Truppel, W. Clustering of streaming time series is meaningless. *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 56 – 65, 2003.

[Mac67] MacQueen, J.B. Some methods for classification and analysis of multivariate observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1:281 – 297, 1967.

[Mar03] Marx, Z. & Dagan, I. & Buhmann, J.M. & Shamir, E. Coupled clustering: a method for detecting structural correspondence. *The Journal of Machine Learning Research*, 3:747 – 780, March 2003.

[Mic02] Michael, C.C. & Ghosh, A. Simple, state-based approaches to program-based anomaly detection. *ACM Transactions on Information and System Security*, 5(3):203 – 237, August 2002.

[Mil03] Miloslavsky, M. & van der Laan, M.J. Fitting of mixtures with unspecified number of components using cross validation distance estimate. *Journal of Computational Statistics & Data Analysis*, 41(3 - 4):413 – 428, January 2003.

[Min] Min, S.L. & Choi, J.D. An efficitent cache-based access anomaly detection scheme.

[Mon05] Montgomery, E.B. & Huanga, H. & Assadi, A. Unsupervised clustering algorithm for n-dimensional data. *Journal of Neuroscience Methods*, 144(1):19 – 24, May 2005.

[Pei01] Pei, J. & Han, J. & Mortazavi-Asl, B. & Pinto, H. & Chen, Q & Dayal, U. & Hsu, M. Prefixspan: Mining sequential patterns efficiently by

prefix-projected pattern growth. *Proceedings of the 17th International Conference on Data Engineering*, page 215, 2001.

[Puz00] Puzicha, J. & Hofmann, T. & Buhmann, J.M. Histogram clustering for unsupervised segmentation and image retrieval. *Journal of Pattern Recognition*, 33(4):617 – 634, April 2000.

[Rab89] Rabiner, L.R. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 – 286, February 1989.

[Sam05] Same, A. & Ambroise, C. & Govaert, G. A classification em algorithm for binned data. *Journal of Computational Statistics & Data Analysis*, October 2005.

[Smy00] Smyth, P. Model selection for probabilistic clustering using cross-validated likelihood. *Journal of Statistics and Computing*, 10(1):63 – 72, January 2000.

[Sod03] Soderlind, G. Digital filters in adaptive time-stepping. *ACM Transactions on Mathematical Software*, 29(1), March 2003.

[Ste05] Steinward, I. & Hush, D. & Scovel, C. A classification framework for anomaly detection. *Journal of Machine Learning Research*, 6(0):211 – 232, March 2005.

[Sur05] Surdeanu, M. & Turmo, J. & Ageno, A. A hybrid unsupervised approach for document clustering. *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 685 – 690, 2005.

[Tas05] Tasoulis, D.K. & Vrahatis, M.N. Unsupervised clustering on dynamic databases. *Pattern Recognition Letters*, 26(13):2116 – 2127, October 2005.

[Vil06] Villmann, Th. & Schleif, F. & Hammer, B. Comparison of relevance learning vector quantization with other metric adaptive classification methods. *Journal of Neural Networks*, 19(5):610 – 622, June 2006.

[Yam00] Yamanishi, K. & Takeuchi, J.I. & Williams, G. & Milne, P. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 6:320 – 324, March 2000.