Experiments with Data Mining in Enterprise Management

A. Knobbe, D. van der Wallen
Syllogic B.V.
Postbus 2729
3800 GG Amersfoort
The Netherlands
{a.knobbe,d.van.der.wallen}@syllogic.com

Lundy Lewis
Cabletron Systems Inc.
40 Continental Blvd.
Merrimack, New Hampshire
USA
lewis@ctron.com

Abstract

This paper describes experiments in applying data mining techniques to historical data collected by network monitoring agents. Large amounts of performance data, including network, system, and application performance data, are collected and stored by monitoring agents. Data mining algorithms analyze the data and codify it into usable knowledge. We show, via experiments, that the knowledge contains useful and unexpected suggestions for improving the effectiveness of business processes and for reducing management support effort. Four experiments are discussed: three preliminary lab experiments and one large, real-world experiment at a major airline company.

Key Words

Case Studies and Experiences, Management Tools and Applications

1. Introduction

As enterprise networks become larger and increasingly heterogeneous, it becomes difficult to manage the complex interaction between different components such as hosts, devices, databases, and applications. Each of these components can affect the operation of the other components, and ultimately affect the performance of the services and business processes they support.

Some events that cause a degradation of the performance in a computer application or business process are simple and can easily be solved or prevented by a human troubleshooter. However, with the increasing complexity of modern computer systems, it becomes quite hard to pinpoint the exact cause of some degradations. In many cases bad performance is not caused by a single, sudden event, but by a subtle interplay between several performance parameters that causes a gradual change in performance. Thus, application management becomes a complex diagnostic problem instead of a set of simple reactive operations.

Although it is hard for human beings to get clear insight into the dynamics and mutual influences among the components of a complex application, it is relatively

easy to monitor the different components in a computer system with current monitoring packages, and to collect raw performance data into a data repository. The Spectrum Enterprise Management Platform and the Spectrum Data Warehouse are good examples of this. Now, given the availability of such historical data, the application of Data Mining techniques in order to extrapolate usable management information is an obvious step [2, 6, 7 10, 16, 17, 18].

The goal of data mining in enterprise management is to transform large amounts of raw data into information, or knowledge, that can be comprehended and used by network administrators. For example, such knowledge may take the form of discovering cause-and-effect relationships among several components in a system, or being able to discover network parameters which distinguish a healthy service from an unhealthy service.

Below we discuss and evaluate results from four experiments using data mining to solve problems in enterprise management.

2. Terms and Concepts

In this section we lay groundwork, describing the notions of monitoring agents, knowledge representation, and domain knowledge. In addition, we describe three data mining tools.

2.1 Monitoring Agents

The first requirement for a data mining application is to collect and store data that describes the state of a system at regular intervals. The data can include configuration data, events and alarms, and, importantly, performance data. Below is a sample of commercial monitoring agents that may be used for this purpose.

- Network Agents have a focused view of the connection nodes in the network infrastructure, for example bridges, hubs, and routers. Service parameters typically include port-level statistics. Good examples in the industry include Cisco's CiscoWorks and Bay Network's Optivity.
- Traffic Agents have a focused view of the traffic that flows over transmission media in the network infrastructure. Examples of service parameters include bytes over source/destination pairs and protocol categories thereof. An example in the industry is NDG Software's Programmable RMON II+.
- System Agents have a focused view of the systems that live in the enterprise. Typically, these agents reside on the system, read the system log files, and perform system queries to gather statistics. Good examples are Metrix WinWatch, BMC Patrol, Tivoli TME, and CA TNG. Service parameters include CPU usage, disk partition capacities, and login records.
- Application Agents have a focused view of business applications that live in the enterprise. These agents also reside on the system that hosts the application. Good examples are Optimal Application Expert, Metrix WinWatch, Platinum ServerVision, and BMC Patrol. Some applications offer agents that provide indices into their own performance levels. Service parameters include thread distribution, CPU usage per application, and file/disk capacity per application.

- Special-Purpose Agents can be built to monitor parameters that are not covered by any of the above. A good example is an agent whose purpose is to issue a synthetic query from point A to point B and (optionally) back to point A to measure reliability and response time of an application. Note that the synthetic query is representative of authentic application queries. An example is an email agent that monitors the response time and jitter of emails from one user domain to another. A good example in the industry is Optimal's Application Expert, which offers agents specifically designed for application monitoring and control.
- Enterprise Agents have a wide-angle view of the enterprise infrastructure, including connection nodes, systems, and applications that live in the enterprise. These agents are also cognizant of relations among the components at various levels of abstraction, and are able to reason about events that issue from multiple enterprise components. This is called event correlation or alarm roll-up. Service parameters that are accessible by enterprise agents are numerous, including router and hub statistics, ATM services, frame relay services, and link bandwidth usage. Examples in the industry are the Cabletron SpectroServer, HP OpenView server, and IBM Netview server.

The data collected by a set of agents is organized into a time-ordered set of *parameter vectors.* The set of monitoring agents produce a vector of measurements periodically to reflect the state of the system at a particular time increment, or over the interval between the previous and current measurement.

For example, consider Service Level Agreements (SLAs). If we consider a month's worth of parameter vectors at ten minute intervals, where some particular parameter has been designated as the service level metric, then we can apply data mining algorithms to discover how other parameters influence the behavior of the service level metric.

2.2 Knowledge Representation: Propositional and First-Order Logic

We distinguish between two primary knowledge representations: propositional logic and first-order logic (a.k.a. predicate logic, or quantifier logic) [19].

In propositional logic, the unit of what we can say is a whole sentence, and we may use the usual Boolean operators to create complex sentences. In first-order logic, our language is more fine-grained, and thus more expressive. Our units of description are objects and predicates, and we are allowed to use universal and existential quantifiers to talk about sets of objects.

For example, consider the simple sentence "The cat is on the mat." In propositional logic this fact is represented by single variable *P*, whereas in first-order logic it may be represented as *Mcm* (where *M* stands for the predicate "is on" and *c* and *m* stand for objects "cat" and "mat"). Further, we may say things in first-order logic such as "All cats are on the mat" -- $(x)\ (Cx > Mxm)$ -- which cannot be expressed in propositional logic.

Most data mining algorithms discover propositional knowledge, although recent algorithms learn first-order knowledge. Clearly, the latter is preferable because it is more general and thus has more predictive power.

Decision Tree Induction Algorithms

Decision tree induction algorithms produce a hierarchical organization of the dependencies among historical data. By starting at the root of the tree we can examine important dependencies, and go into further detail by proceeding towards the leaves of the tree. Popular algorithms of this kind are ID3 and C4.5 [14].

Decision trees suffer from the fact that particular dependencies among parameters may be overlooked. Suppose that the behavior of one parameter describes the behavior of a target parameter. Then the remaining parameters may be overshadowed although they also affect the behavior of the target [9, 10]. We will see an example of the overshadowing phenomenon later in our experiments.

Top N Algorithms

The overshadowing problem is quite clear when two parameters are considered of which one is some increasing function of the other. Unfortunately this is often the case in the networking domain. Therefore it may be useful in most cases to produce one primary decision tree, and also a ranked list of the top parameters that are possible explanations of the target parameter behavior.

Rule Induction Algorithms

Rule induction algorithms may be used to produce overlapping rules that will show multiple dependencies between a target parameter and remaining parameters. Also many parameters can be expressed as arithmetic functions of other parameters. Roughly linear dependencies between parameters are quite common. Simple statistical techniques such as correlation and linear regression can also be very effective to produce a top N.

Inductive Logic Programming

A shortcoming of the three data mining algorithms described above is the propositional nature of the inferred knowledge – for example: *If parameter X on machine Y > 5 then Performance is low.* The statement is indeed useful; however, one is inclined to ask further questions: Are there other machines for which this rule holds also? How many classes of machines are there to which this rule is applicable? Do there exist instances of such classes in my enterprise?

Inductive logic programming (ILP) offers an approach to data mining that produces general knowledge expressed in terms of first-order logic [4, 11, 12, 13, 15]. ILP incorporates special knowledge in the form of a logic program which specifies the links and relationships already known to be in the domain of the application. In our experiments this background knowledge consists of a component-wise relational model and a hierarchical decomposition of components into subcomponents.

Domain knowledge is used by ILP to infer more general knowledge. The algorithms can discover propositional knowledge such as *diskio of HDisk1 on Luxor,* and can also produce knowledge at higher levels of abstraction, e.g. *diskio of a disk on an AIXServer.* Such results are more interesting for system administrators,

as they appear more like genuine knowledge than do individual propositions. In addition, the knowledge enjoys more predictive power.

Domain Knowledge

The domain knowledge that is used in our experiments consists of a collection of relations between different components in the system. For example, the hierarchical description of components consists of the following logical facts: an AIX4-r1server is an AIXserver is a UNIX server. The description of the model consists of relations between components. The relations between components are for example: a disk is a part of a machine, and a machine is part of an application.

2.3 Data Mining Tools

In our experiments, we have used three data mining tools to analyse the data collected by monitoring agents: the Adaptive System Management Tool from Syllogic (Netherlands), Progol from the University of York (UK), and Tilde from the University of Leuven (Belgium). We describe these briefly below.

Adaptive System Management Tool

The Adaptive System Management (ASM) Tool, developed at Syllogic in the Netherlands, is an integrated toolbox that consists of separate modules for data collection, data storage, system modelling, data mining, and presentation. The mining module consists of the three propositional techniques described in Section 2.2. The results of these techniques are presented visually to the user. In the system modeling module the analyst defines domain knowledge.

The ASM tool uses a hierarchical architecture for organizing monitoring agents and collecting data. The basic component of the data collection system is a collection object. Each collection object performs the same action: (i) retrieve data, where the source of data is either a system or another collection object, (ii) store the data in memory, file or database and potentially (iii) propagate data to parent collection objects. At the leaves of the tree the collection objects retrieve data by taking measurements (e.g. by executing programs) or reading existing statistics (e.g. SNMP or kernel statistics). The internal nodes of the tree collection objects retrieve data by collecting data from the collection objects that are below them. Finally, the root node stores all data into a repository such as a Data Warehouse.

An advantage of this hierarchical structure of monitoring agents is that one can construct a collection scheme that makes effective use of resources (network, storage, and computational power) during collection, query, and storage phases. Furthermore, by organizing the tree to minimize dependencies, the system can continue to operate when there may be a problem in one or more agents in the system.

Progol

Progol, developed at Oxford University Computing Laboratory, is an ILP system that uses a rule-induction algorithm [13]. It discovers rules that explain as much of the data as possible. The rules may overlap in the data coverage. Progol exploits background domain knowledge, making it possible for specific rules to be

incorporated in later learning, thereby discovering more general rules expressed in first-order logic. It constructs a pruned top-down search which produces guaranteed optimally short clauses. Furthermore, it supports numerical hypotheses using built-in functions as background knowledge.

Tilde

Tilde, developed at the University of Leuven, is an ILP system that builds binary decision trees, having no overlapping coverage [4]. It is derived from Quinlan's propositional C4.5 algorithm, in that Tilde uses the same heuristics and search strategy. However, Tilde extends C4.5 in that it discovers first-order representations. Hence, it has C4.5 capability (when working with binary attributes) as a special case. Because of the use of the divide-and-conquer strategy underlying decision tree technology, Tilde has proven to be very efficient when run on large datasets. Other features of Tilde include the ability to handle numbers and to perform regression.

3. Three Preliminary Lab Experiments

In this Section we describe briefly three preliminary experiments on real life data sets. In the first two experiments we relied upon learned propositional structures provided in the ASM tool set. For the third experiment we used similar data mining algorithms in SAS. More detail can be found in the references. These three experiments layed the groundwork for a real-world trial, which we descibe in Section 4 in some detail.

3.1 File System Performance

The aim of this experiment was to discover the parameters that influence the access time to files in a file-system distributed over several hosts [10]. To do this, a data set was produce by periodically executing several UNIX commands, with an interval of 30 minutes, and measuring the response-time. The result was a data set of 5052 records with the following attributes:

Weekday:	Sunday, … , Saturday
Directory:	name of directory
Local:	location of directory
Filehost:	name of host
Dirsize:	total size in kB
CPU_usage:	load on local host
Response:	response-time below 2 seconds

The parameter Response was the target. All other parameters were potential causes except Filehost, which depends on Directory.

Figure 1 shows a part of a decision tree for all general or specific, potential primary causes. The numbers below each node in the tree indicate the total number of cases and the percentage of cases with a bad response, respectively. Clearly the name of the directory almost completely determines the acceptability of the response time. However, the tree does not show any information about any of the

other parameters because they are overshadowed by Directory. If the analysis is restricted to the general parameters, the tree in Figure 2 is obtained. Clearly the response is never acceptable (the leaf is 100%) if the directory is remote and the directory-size exceeds 2652 KB.

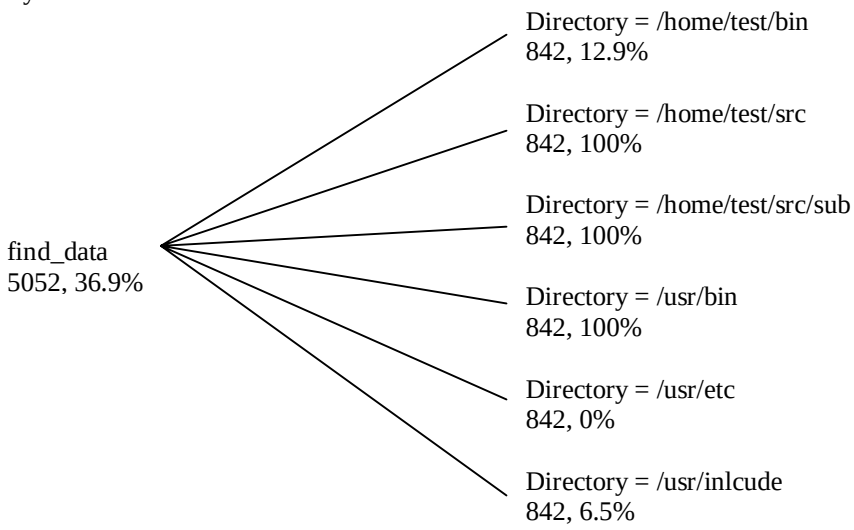

Figure 1: Decision tree with overshadowing

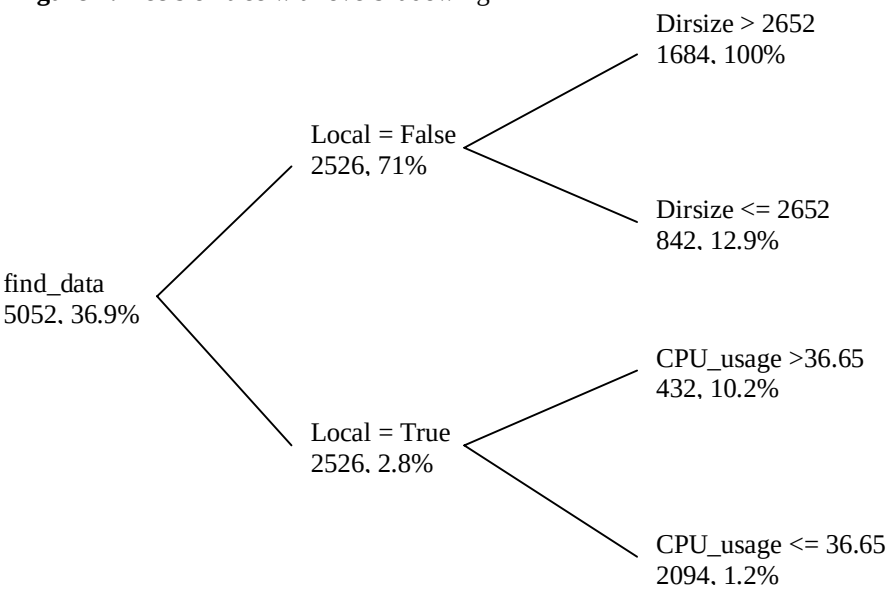

Figure 2: Decision tree without overshadowing

3.2 Network Bottlenecks

The data for this experiment describes the load on 16 subnets connected by a single router [10]. Over a period of 18 weeks a total of 16849 measurements were done on each of the subnets. The following parameters were monitored for each subnet. Each measurement represents the state over a 10-minute period. Along with these parameters, the day of the week and the time of the day were measured.

LoadN:	percentage of bandwidth utilization on subnet N
PktsN:	packet throughput on subnet N
CollN:	number of collisions on subnet N

This data was analyzed in an attempt to pinpoint a problem causing complaints about occasional poor performance on subnet 5. At times the load on subnet five was too high for people to work on. The problem would disappear after an hour but a probable cause was not found. A preliminary investigation showed that users started complaining whenever the load on subnet 5 started creeping above 35%, so a single target was defined that is true iff Load_5 is above 35%. Before the analysis was started, the parameters Pkts_5 and Collision_5 were removed from the data set because they are tied directly to Load_5.

The decision tree in Figure 3 shows part of the dependencies between the target parameter and the remaining parameter. Clearly the performance on subnet 5 is unacceptable in most cases that the load on subnet 9 was high.

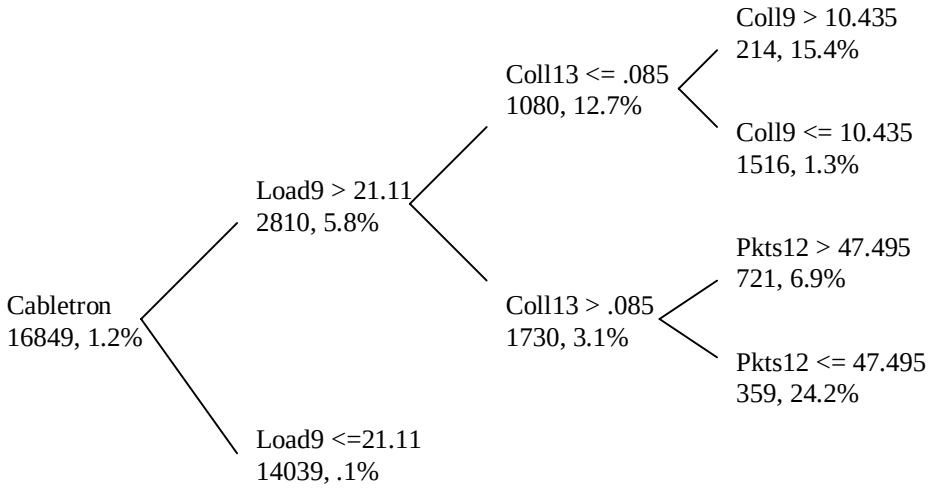

Figure 3: Decision tree for the network bottleneck problem

The dependency between subnet 5 and 9 seems to suggest that there is some sort of client/server communication going on between the two subnets. If such a communication actually is the main cause for the high load on subnet 5, there should be a roughly linear relation between the number of packets or the load on the

two subnets. To test this hypothesis, a correlation analysis was done on all pairs (Load_5, Load_i), resulting in the following top three absolute correlations:

Load5	Load9	.788
Load5	Load1	.131
Load5	Load4	.105

3.3 Discovering System Behavior Models

This experiment was designed to discover a state transition graph (STG) that models the behavior of a network [7]. If the learned STG were accurate and reliable, then it could be used for proactive and reactive fault management, prediction of network behavior, and automated monitoring and control.

The same numeric data set was used as that described in Section 3.2 above; however the components of the learned STG were qualitative, including the states, input events, and state transitions. The questions we tried to uncover in this experiment were (i) which mining methods provide better results for this problem, (ii) how much and what kind of data do we need to extrapolate a useful behavior model, and (iii) should we derive one global behavior model, or should we derive some number of local behavior models. As this is more-or-less a lab experiment which is rather involved, we refer the reader to the details described in [7].

4. A Large, Real-World Experiment

This experiment was performed at a large airline company in the Netherlands. We monitored a spare part tracking and tracing appplication (TTA) for aircraft. The TTA uses several IBM AIX servers, an Oracle database, and Windows PC clients in Amsterdam, Singapore, and New York. In total, monitoring agents were put into place that collected values on 250 parameters at regular intervals. Examples of some of the parameter types are *cpu load, free memory, database reads* and *nfs activity*. The agents performed a read every 15 minutes and stored the values in a database. The TTA was monitored for 2 months, resulting in a table of 3500 time slices of 250 parameters.

The performance was measured by simulating a task of the application, viz. querying a database, and recording the access time of this database query. The performance measure was declared as the pivotal measure in a Service Level Agreement (SLA) between the IT Department and the users of the TTA. All parameters, including the *SLA* parameter, were measured at the same interval. The determinator of good and bad performance of the TTA is governed by the test *database access time* < *3*. This means that a TTA user should never have to wait more than three seconds for the answer when performing this specific task. This task was agreed to be typical for TTA users.

A model of the TTA and the underlying supporting systems was created. The model consists of about 140 components, such as servers, printers, disks and databases. All components that are monitored are in the model as well as the known relations between the components. The relations are one-to-many, part-of relations.

Results of the Decision Tree Algorithm

The tree in Figure 4 shows the decision tree induced from the complete dataset. The most important parameter is *paging space used on server 11*. The tree states that a high value for the parameter *paging space used* is the main indicator for bad performance. The amount of used paging space can not be influenced directly, but can be reduced by increasing the amount of physical memory, or limiting the number of applications that run on this server. The next split on *CPU idle time on server 11* gives additional evidence for the fact that server 11 needs additional hardware or less intensive usage.

The parameter *filetime* measures the delay on scheduled jobs. The application updates requests from a mainframe at fixed times. The requests are sent (in batch) from the mainframe to the UNIX-server server 11, and then processed by the application. The split suggests that if the mainframe sends the file more than 2.5 minutes late, the performances drops. This can have multiple causes. Firstly, the network could be down causing the mainframe to fail when trying to send the requests and at the same time causing the performance-measure to time out because the database query is performed over the network. Secondly, the application wastes CPU cycles trying to retrieve a file which is not yet there, because the mainframe application did not yet put it there. This happened in more than 50% of the cases. So, this split indicates that the way the application receives mainframe requests should be improved.

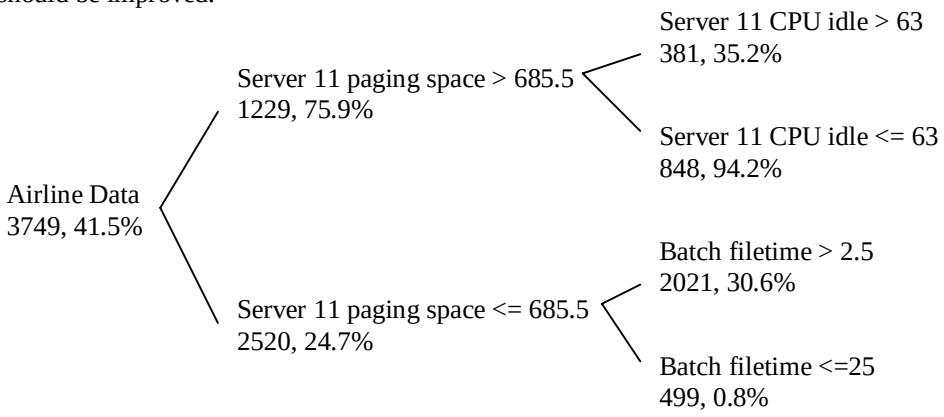

Figure 4: Results of the decision tree algorithm

Results of the Top N Algorithm

The parameters that influence performance were discovered by the Top N algorithm. The resulting top 3 are displayed as follows:

Server 11 paging space > 685.5 MB
Client 6 ping time > 258.5 millisec.
Server 5 CPU idle < 74.5 %

The attribute *paging space used* is in agreement with the results in the previous section. The attribute *client ping time* is the ping time to a foreign router. It is clear that if this ping time exceeds 258.5 milliseconds the performance for foreign users is bad. A system manager of the airline company reasoned that these high ping times correspond to foreign users loading a complete table from the database to their client. This table could be very big and the network connections to foreign countries have a narrow bandwidth. So this situation caused both the network as well as the application to be very busy for some time. Unfortunately, fixing this bottle-neck is very expensive: buy more bandwidth or redesign and reimplement a part of the TTA to reduce network traffic.

The presence of the attribute *CPU Idle time on server 5* is a direct indication that this machine is a bottle-neck for the performance of the TTA.

Results of the Rule Induction Algorthm

With the rule induction algorithm we induced several types of rules. Simple rules that contain one parameter as a condition and one as a conclusion are presented in an association matrix. We found several surprising relations between individual components of the system. A striking example is the correlation of two disk IO monitors: the correlation between the disk IO of a disk in server 5 and a disk in server 25. After studying the data more closely it was concluded that these disks exhibited almost complete disjunct operation.

The rule induction algorithm also discovered rules involving (multiple) parameters as conditions that affect TTA performance. We found, for example, the following rules:

low freespace on /var/tmp on server 5 -> low performance
server 25 high paging space $\wedge$ server 25 high ftp connections -> low performance

Results of Algorithms That Learn First-Order Structures

This section describes *ILP* experiments. To compensate for the growth of search space by the addition of the model information and using a first order modeling technique, we preprocessed the data set into binary parameters incorporating only values that are higher or lower than average $\pm$ 3 times the standard deviation. This reduces the data set size (leaving out all "normal" values) and simplifying the search, since only binary parameters are used. The loss of information in this preprocessing step has been taken as a simplifying assumption. The main goal of applying first order techniques was to give a proof of concept of the usability of ILP. Using all values without reducing the search space may have produced better results but also would have dramatically increased the analysis time.

Progol

Progol was used to induce rules that incorporated the relevant model information. Progol generated the two rules displayed below.

Performance is low <- $\exists$x class_id(x, # requests in queue) & high(x)
Performance is low <- $\exists$x class_id(x, NFS server) & high(x)

Because there is only one parameter of type *number of requests in queue*, this rule is essentially propositional. The queue contains the requests that still have to be dealt with. This means the application cannot handle all incoming requests, or perhaps that too many users use the application at the same time.

The second rule means that if any parameter of type *NFS server* on any server in the model is high the performance will be low in the coming period. Although there are just six parameters that have type *NFS server*, the second rule is very interesting because it gives a more general description of the problem, which is that on server 11, the network load and the usage of this server is too high. Note that several specific correlations that we found with the propositional algorithms are expressed here in one understandable rule.

Tilde

Tilde was presented with the same preprocessed data as Progol. Tilde induced the tree in Figure 5.

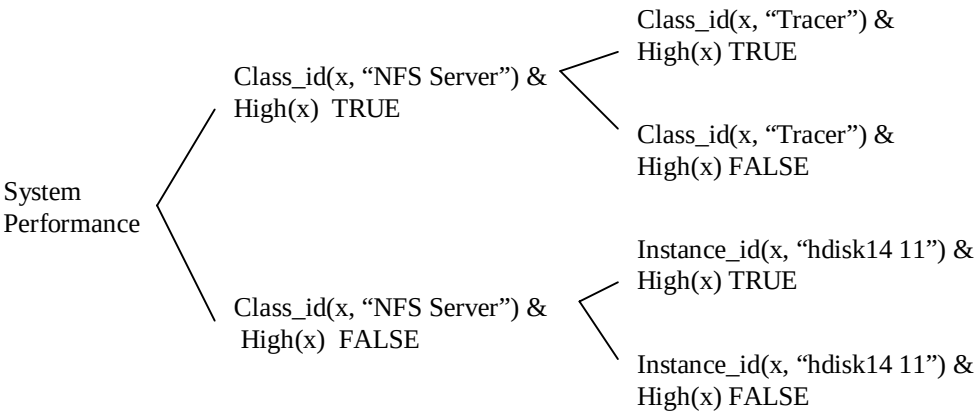

Figure 5: Tilde Decision Tree

The first split in the tree is equivalent to the following rule: If there is a parameter with class NFS Server that has value high then performance is low.

This is exactly the same result as found with Progol. High values of the *NFS Server* class have the greatest impact on the performance. The next split is on high values of class *Trace*. These parameters are the different performance measures defined for this application. So, this split tells us that the different performance measures are similar in the case of high values of class *NFS server*. If there aren't high values of class *NFS server* Tilde splits on high parameter values for the instance *Hdisk14 server 11*. A parameter instance is a collection of parameters on a specific component, so this comes down to the fact that high usage of disk 14 on server 11 is causing performance decrease. Further investigation shows that this holds for almost all disks of server 11.

Recall that from the propositional experiments we concluded that memory problems or overloading of server 11 was the main problem of the TTA. Here we see something similar. When NFS activity (on server 11) is low, high disk activity on server 11 is the main bottleneck. It is fairly easy to identify this situation as

swapping. The machine has a low CPU load, low network activity but is only swapping memory causing high disk activity. Again, the conclusion is that server 11 needs more memory or the number of applications on this server should be decreased.

5. Discussion: Data Mining in Enterprise Management

Our experiments in applying data mining methods to archived network data have been encouraging, and now we have been able to produce results that are meaningful at a customer site.

Our task at the customer site was a common one: to understand the causes that effect the behavior of SLA performance metrics. Clearly, if such causes can be discovered, then one is halfway towards making performance better and thus more likely to meet SLA commitments. Below we suggest a number of other tasks that we think are amenable to data mining methods, each one of which deserves some old-fashioned brainstorming, discussion, and implementation. Our immediate goal at this juncture is to incorporate these methods into commercial product, where little to no technical expertise in data mining or machine learning methods is needed in order to get useful results.

- Identify the common characteristics of applications or business processes that appear to perform well, in contrast to those which perform poorly.
- Predict which applications or processes are likely to perform poorly in the future.
- Identify security breaches in network usage.
- Understand what networking services and applications are commonly used together.
- Reveal the difference between an application or process this month versus last month.
- Identify the causes of faults in historical data sets [17].
- Evaluate new network configurations [17].

6. Summary and Conclusion

The concepts and experiments presented in this paper demonstrate that the application of Data Mining techniques to the domain of enterprise management can offer insight into the dynamics of the system at hand. In the experiments we found several unexpected and real problems with the applications and business processes under study. We have seen that, although propositional techniques are very useful, first-order techniques are more comprehensible and have more predictive power. Further work should be done in the area of an efficient implementation of first order techniques. Also further research should be done to determine domain specific properties that can be used to improve the efficiency and quality of the results. We provided brief discussion of three lab experiments, and a large-scale implementation of data mining at a customer site. Finally, we suggested other areas that would be amenable to data mining methods.

References

[1]. Adriaans, P.W. and Zantinge, R. *Data mining.* Addison-Wesley, 1996.

[2]. Adriaans, P.W. *Adaptive System Management*, in Advances in Applied Ergonomics, proceedings ICAE'96, Istanbul, 1996.

[3]. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A. 1996. *Fast discovery of association rules*, in [5].

[4]. Blockeel, H., De Raedt, L. *Top-down induction of logical decision trees.* Unpublished. 1997.

[5]. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. *Advances in Knowledge Discovery and Data Mining*, AAAI Press/MIT Press, 1996.

[6]. Herlaar, L. *Diagnosing Performane of relational database managament systems*, technical report, Utrecht University, 1995.

[7]. Ibraheem, S., Kokar, M., Lewis, L. *Capturing a qualitative model of network performance and predictive behavior*, Journal of Network and System Management. In press.

[8]. Knobbe, A.J., Adriaans, P.W. *Analysing binary associations*, in Proceedings KDD '96.

[9]. Knobbe, A.J., Den Heijer, E., Waldron, R. *A practical view on data mining*, in Proceedings PAKM '96.

[10]. Knobbe, A.J. *Data mining for adpative system management*, in Proceedings of PADD '97

[11]. Lavrač N., Džeroski S. *Inductive Logic Programming,Techniques and Applications*, Hellis Horwood, 1994.

[12]. Lavrač N., Džeroski S. *Inductive Logic Programming, proceedings ILP-97*, Springer, 1997.

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

[14]. Quinlan, J.R. *C4.5: Programs for Machine Learning*, Morgan Kaufman, 1992.

[15]. de Raedt, L. (Ed.) *Advances in Inductive Logic Programming*, IOS Press, 1996.

[16]. Zantinge, R. and Adriaans, P.W. *Managing Client/Server.* Addison-Wesley, 1996.

[17]. Venter, F. J. The use of lattice-based knowledge discovery to determine dependencies between network management parameters. Masters Thesis, University of Pretoria, South Africa. October 1997.

[18]. Lewis, L. and Noushin, A. Data mining in large network archival databases. Technical Note ctron-lml-95-02. Cabletron Systems. 1995.

[19]. Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall. 1995.