UNIVERSITY OF AGDER

Online Failure Prediction in UNIX Systems

Peter Allan Svendsen

Supervisor
Associate professor Ole-Christoffer Granmo

This Master's Thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

University of Agder, 2011
Faculty of Engineering and Science
Department of ICT

Abstract

This thesis investigates the possibility of enhancing an existing performance monitoring system for UNIX servers, by adding the capability of predicting upcoming failures, using generic UNIX operating system performance metrics like used server memory, CPU utilization, I/O traffic etc. as input data for machine learning and pattern recognition. In this thesis we survey possible research methods based on input data they process, and propose a novel approach for symptom based failure predicting. In order to make a generic solution that can be used on any UNIX computer, we have only used open source software. We evaluate the classifiers Naive Bayes and Logistic Regression with input data in both standard and vectorized format. Furthermore we use the search algorithm Forward stepwise selection to find an optimal generic set of variables (features) that improves the quality of the classification. Our empirical testing demonstrates that our proposed method is capable of predicting symptoms with high overall accuracy, but the uncertain quality of the monitored performance data used as input makes it difficult to ascertain if the symptoms are actually failures. Applying the search algorithm for feature selection and vectorizing the input data set we improved the time for classification with an order of magnitude. In our opinion the proposed technique for online failure prediction will benefit to applications concerning performance monitoring and contribute to the research field of online failure prediction with new insight.

Preface

This master thesis is submitted in partial fulfillment of the requirements for the degree Master of Science in Information and Communication Technology at the University of Agder, Faculty of Engineering and Science. This work is supported by Skatteetatens IT og Servicepartner in Grimstad where I am employed. The project's supervisors have been associate professor Ole-Christoffer Granmo and phd. student Anis Yazidi from University of Agder. I'd like to thank SITS for making it possible for me to complete this study, Ole-Christoffer and Aniz Yiszaldi for their splendid supervision and my wife Connie for all the help and support. Last but not least I'd like to thank my fellow colleagues for relieving my workload, family and friends for all support and patience.

Grimstad, December 16, 2011
Peter Allan Svendsen

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this work we investigate the possibility of enhancing an existing performance monitoring system for UNIX servers used at Skatteetatens IT og Servicepartner in Grimstad, by adding the capability of predicting the future possibility for a server to fail. We examine related work and outline all possible methods for online failure prediction. We use generic UNIX operating system performance metrics like used server memory, CPU utilization, I/O traffic etc. as input data for the machine learning and pattern recognition to detect failures. Gathering the performance information for predictions with operating system commands that are generic for all UNIX genres gives the advantage of making the solution portable to any UNIX computer and without any extra expenses for software. We make a model that uses a script with the "AWK" command to prepare the performance data for the classifiers. Evaluating different classifiers we find that Naive Bayes and Logistic Regression are well suited for this task The model is then used to analyze the performance metrics in order to optimize the performance and accuracy for the classifiers. Finally we evaluate the classifiers and select Naive Bayes for the task.
In section 1.1 we show state of the art research for online failure prediction. In the following section 1.2 we define "Online failure prediction", before we present our motivation in Section 1.3. Section 1.4 contains the full thesis definition. In Section 1.5 we state the key assumptions and limitations. A presentation of the contributions are given in Section 1.6. We end this chapter with a section describing the intended target audience of this thesis in Section 1.7, and present an overview of the thesis layout in Section 1.8.

1.1 State of the art

In this section, we give a brief overview of some of the latest work that has been carried out in the field of online failure prediction. We give examples from the all three main categories defined by Salfner et al. [10] for online prediction according to the type of input data that they process. At the end of this section we mention the importance of feature selection (variables) and finally explain the novelty of our research.

Failure tracking evaluate the times of previous failure occurrence. Due to sharing of resources, system failures can occur close together either in time or in space. It has been observed, that failures occur in clusters in a temporal as well as in a spatial sense. (For further explanation see chapter 2.1.1)

Liang et al. [21] in 2006 used such an approach to predict failures of IBM's BlueGene/L from event logs containing reliability, availability and serviceability data. The key to their approach is data preprocessing, employing first a categorization and then temporal and spatial compression: Temporal compression combines all events at a single location occurring with inter-event times lower than a determined threshold, and spatial compression combines all messages that refer to the same location within a given time window. Prediction uses data from temporal compression, if a failure of type application I/O or network appears, it is very likely that another failure will follow shortly. If spatial compression suggests that some components have reported more events than others, it is very likely that additional failures will occur at that location.

Detected error reporting approaches evaluate the detection of errors in log files that have not yet evolved to become a failure. (For further explanation see chapter 2.1.3)

Lal and Choi in 1998 [24] used log files as input data to detect errors and failures in a UNIX server. They show plots and histograms of errors occurring in a UNIX server and propose to aggregate errors in an approach similar to tupling. They state that the frequency of clustered error occurrence indicates an upcoming failure. Furthermore, they show histograms of error occurrence frequency over time before failure.

Later, Leangsuksun et al. in 2004 [19] presented a study where hardware sensors measurements such as fan-speed, temperature, etc. are aggregated using several thresholds to generate error events with several levels of criticism. These events are analyzed in order to eventually generate a failure warning that can be processed

by other modules. The study was carried out on data from a high-availability high-performance Linux cluster.

Salfner and Malek in 2007 [26] use error log files with data from a commercial telecommunication system as input data to predict performance failures. They state that. “Error-logs are a fruitful source of information both for diagnosis as well as for proactive fault-handling however elaborate data preparation is necessary to filter out valuable pieces of information”. They use well known techniques, and proposed three algorithms: (a) assignment of error IDs to error messages based on Levenshtein’s edit distance, (b) a clustering approach to group similar error sequences, and (c) a statistical noise-filtering algorithm. Two Hidden Semi Markov Models are trained: One from failure and one from non-failure sequences. HSMMs extend standard hidden Markov models by defining cumulative probability distributions in order to specify the duration of state transitions. With this approach, the two HSMMs learn to identify the specifics of failure and non-failure sequences. This paper shows that data preparation is an important step to achieve accurate error-based online failure prediction. Analysis show good prediction performance on field data of an industrial telecommunication system.

Symptom monitoring assumes that symptoms are side-effects of errors. Thus the approaches evaluate monitoring data reflecting symptoms (side-effects) of errors. (For further explanation see chapter 2.1.2)

In 2001 Hamerly and Elkan [12] use Naive Bayes classifiers for failure prediction in hard-disk drive failure prediction. Input data used is time-driven SMART (self-monitoring and reporting technology) attributes that comprise for example spin-up time, power-on hours and counts for seek errors and CRC errors. The authors propose two Bayesian methods for the prediction of hard disk drive failures: an anomaly detection algorithm that utilizes a mixture of naive Bayes sub-models and a naive Bayes classifier that is trained by a supervised learning method. The first method builds a probability model only for drives behaving normal. The Naive Bayes Model is trained by using Expectation-Maximization and is hence called NBEM. The second method is computing conditional probabilities for SMART values belonging to the failure or non-failure class. Both methods are tested on real world data from 1936 hard disk drives. The predictive accuracy of both algorithms is far higher than the accuracy of threshold methods used in the disk drive industry

at the time of 2001.

The Naive Bayes classification requires that input variables take on discrete values. Therefore, monitoring values are often assigned to a finite number of bins as in Hamerly and Elkan [12]. This can lead to poor assignment of features to bins, if monitoring values are close to a bin's border. Fuzzy classification addresses this problem by using probabilistic class membership. Turnbull and Alldrin [28] in 2003 use Radial Basis Functions networks (RBFN) to classify monitoring values of hardware sensors such as temperatures and voltages on motherboards. More specifically, all N monitoring values occurring within a data window are represented as a feature vector which is then classified to belong to a failure-prone or non failure-prone sequence using RBFNs. Experiments were conducted on a server with 18 hot-swappable system boards with four processors, each. The authors achieve good results, but failures and non-failures were equally probable in the data set.

Other researchers use time series analysis with regression in their approach e.g. Castelli et al. [5] in 2001 say that IBM has implemented a curve fitting algorithm for the xSeries Software Rejuvenation Agent. Several types of curves are fit to the measurement data and a model-selection criterion is applied in order to choose the best curve. Prediction is again accomplished by extrapolating the curve. Cheng et al. [6] in 2005 present a two step approach for failure prediction in a high availability cluster system. Failure prediction is accomplished in two stages: first, a health index $\sum 0 , 1]$ is computed using fuzzy logic, and in case of a detected "sick" state of a node, a linear function is mapped to the monitored values of the resource in order to estimate mean time to resource exhaustion. The authors also reference a technique called "prediction interval" to compute a lower and upper bound for time to resource exhaustion. The fuzzy logic assignment of the health index is based on "processor time", "privileged time", "pool non-paged bytes", and "available Mbytes".

Regression is used by several, for instance: Li et al. [20] in 2002 use a stochastic model and collect various parameters such as used swap space from an Apache web server to build an auto regressive model with auxiliary input (ARX) that predict further progression of system resources utilization. Failures are predicted by estimating resource exhaustion times. They compared their method to Castelli et al. [5] and showed that on their data set, ARX modeling resulted in much more accurate predictions.

Andrzejak and Silva [4] in 2007 apply deterministic function approximation techniques such as splines to characterize the functional relationships between the target function (the authors use the term “aging indicator”) and “work metrics” as input data. Work metrics are, e.g., the work that has been accomplished since the last restart of the system. Deterministic modeling offers a simple and concise description of system behavior with few parameters. Using work-based input variables rather than time-based variables gives the advantage that the function is not depending on absolute time: For instance, with little load on a server, aging factors accumulate slowly and so does accomplished work whereas in case of high load, both accumulate more quickly. The authors present experiments where performance of an Apache Axis SOAP (Simple Object Access Protocol) server has been modeled as a function of various input data such as requests per second or the percentage of CPU idle time.

Function approximation is one of the predominant applications of machine learning. An example of this is the research where Hoffmann in 2006 [2] developed a failure prediction approach based on universal basis functions (UBF), which is an extension to radial basis functions (RBF) that use a weighted convex combination of two kernel functions instead of a single kernel. UBF approximation has been applied to predict failures of a telecommunication system. The method has also been successfully applied to the prediction of resource consumption in the Apache web server. Where Hoffmann et al. [14] in 2007, conducted a comparative study of several modeling techniques with the goal to predict resource consumption of the Apache web server. The study showed that UBF turned out to yield the best results for free physical memory prediction, while server response times could be predicted best by support vector machines (SVM). The basic idea is to permanently monitor characteristic variables such as workload, number of processes, used I/O bandwidth in a software system, the probability of failure occurrence is assumed to be a function of a selection of these input variables. This functional interrelation is learned from previously recorded measurements by proposing a machine learning approach: universal basis functions. This method uses offline selection of parameters (feature selection) from previously recorded training data and then used it to perform an online prediction of failures. If it exceeds some predefined threshold, a failure warning is raised. After applying a feature selection only two features are used, namely the number of semaphore operations per second and the amount of

allocated kernel memory.

Feature selection is concerned with finding the optimal subset of measurements to speed up and get more accurate predictions. This has proved to be very important in several researches. One of the major findings in Hoffmann et al. [2007][14] is that the issue of choosing a good subset of input variables has a much greater influence on prediction accuracy than the choice of modeling technology. This states that the result might be better if, for example, only workload and free physical memory are taken into account and other measurements such as used swap space are ignored. A typical variable selection algorithms is Forward Stepwise Selection (see, e.g., Hastie et al. [13] , (Chapter 3.4.1), which has been used by Turnbull and Alldrin [28] in 2003. In addition to UBF, Hoffmann [2] has also developed a new algorithm called probabilistic wrapper approach (PWA), which combines probabilistic techniques with forward selection or backward elimination.

In this thesis we use symptom monitoring as input for classifiers. This is in principle the same approach as Hamerly and Elkan [12] where the authors use temperature, spin-up time, power-on hours and counts for seek errors and CRC errors as input for Naive Bayes classifier, to get failure prediction for hardware failures. We propose a solution where the novelty in our approach is using generic UNIX performance metrics like memory use, CPU utilization and I/O traffic as input data for Naive Bayes and Logistic Regression to achieve failure prediction in software. Our solution also use Forward stepwise feature selection in order to improve the failure prediction. In the above description of research done in symptom monitoring we see that there are many of the researches which use similar input as we do, but all of them use other principle approaches like time series analysis with regression or the more predominant function approximation approach to predict failures.

1.2 A definition of Online Failure Prediction

The goal of online failure prediction is to predict the occurrence of failures during run-time based on the current operating system state. To give a more precise definition, the terms “failure” and “online prediction” are defined in separate subsections

1.2.1 Failures

Avizienis and Laprie’s[1] definition of failures is commonly used:

> “A system failure occurs when the delivered service deviates from the specified service, where the service specification is an agreed description of the expected service”.

Said with other words: A failure is a misbehavior that can be observed by the user, which can either be a human or a computer component. Error may appear in a system, but if the system delivers its intended service it is not a failure. In this thesis the intended service is the UNIX operating system.

1.2.2 Online prediction

Online failure prediction is to predict during run-time whether a failure will occur in a short period of time based on an assessment of the monitored current running operating system state.

1.3 Motivation

Computer systems are getting increasingly more complex with growing connectivity, interoperability and they are also changing dynamically. Changes are caused by the mobility of computer devices, changing execution environments, frequent updates and upgrades, configurations, online repairs, the addition and removal of system components and the system/network complexity itself. With the high focus on ICT costs, and the fact that we are getting increasingly more dependent on computer systems, users expect and even demand that computer systems should never fail, still systems are inherently prone to failures. Failures have consequences like high costs for users who may experience loss of service for some period of time or even loss off valuable data. To find solutions to this

challenge we have to ask ourselves what are the reasons for computers systems to fail? There are lots of reason for computer systems to fail:

- hardware problems
- environment problems
- third-party, open-source software, Commercial-Of-The-Shelf (COTS) components
- unproven design
- bad configuration
- growing number of attacks and threats
- misuse
- novice users
- and probably most of the times it is due to software faults

What can be done to improve the failure-prone systems? Because, eventually they will fail again! Building better software is a obvious solution, but it seem like some software developers have failed this task. We can find ways to identify failure-prone situations and react accordingly. But how can we know when a failure is about to happen? The obvious solution is to predict the failure! If a failure can be predicted, preventive action can be taken to reduce consequences of the pending failure. This states that proactive fault management is an effective approach to enhancing reliability. Conventional methods does not consider the actual state of a system and are therefore not capable of reflecting the dynamics of run-time systems and failure of processes. Often a root cause analysis is done when the failure has occurred to find and repair a bug. Root cause analysis are typically useful in design for long-term or average behavior predictions and comparative analysis based on:

- failure rates
- architectural properties
- the number of bugs that have been fixed etc.

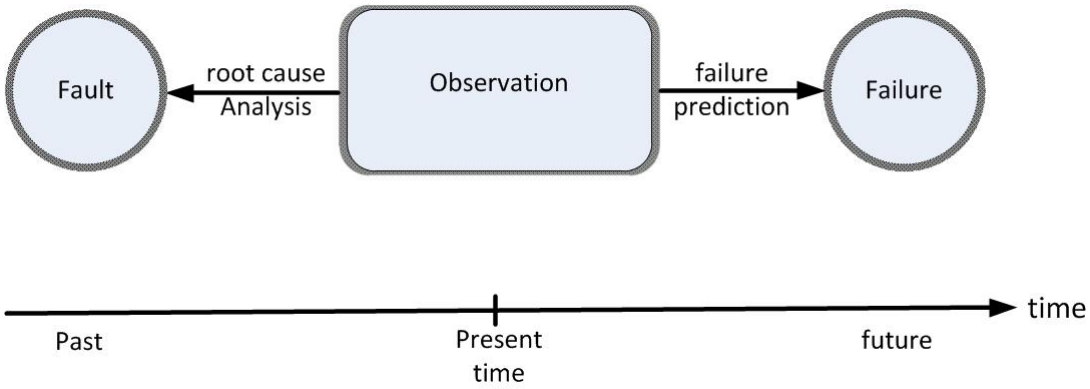


Figure 1.1: Distinction between root cause analysis and failure prediction. Composed from [10]

Short-term predictions on the other hand are made on the basis of run-time monitoring, and we therefore need to develop methods that capture and select the essential data of computer systems. The method must be able to select the most significant variables to failure prediction, which may be only a few of probably many hundreds of variables that could be observed. Next we need to develop a method that interpret the collected data and recognize erroneous system states in order to predict future system failures. Figure 1.1 shows the difference between root cause analysis and online failure prediction

People have always tried to predict the future and today millions of people work with prediction, because we know that if we can predict the future it will give us an advantage. The key to improve proactive fault management is to predict failures before they actually happen. This would give us the advantage to prevent potential disasters or to limit the damage that can be caused by a computer system failure.

Many companies have Service level agreements for different ICT systems depending on how important they are. Often the SLA for an important system says that the system is expected to be available 99 percent of the time in working hours (between 08.00 - 16.00) and if this requirement is not met there will be some penalty. For the operating department in charge of the computer systems it will be a great advantage if they are warned when a failure is imminent. This gives us the necessary time to do maintenance or stop a service so that no data will be lost. In some cases when a failure is imminent a restart of a service that takes 2 minutes can save many hours of restoring from backup. Two minutes downtime can be acceptable in an SLA, but hours will most likely be outside the accepted scope in the SLA. If a prediction can be done correctly then a counter measure can be taken such as:

- Initiation or restart of service.
- manual or automatic configuration
- optimization
- some task for healing
- protection

and thus save companies for large amounts of money.

1.4 Thesis definition

“We will investigate the possibility of using generic UNIX performance metrics like memory, CPU and I/O as input data for "online failure prediction in UNIX systems". We will find a good subset of the input variables to give highest possible prediction accuracy and the best performance. Further we will show that this can be implemented on any UNIX system in a relatively simple manner.

1.5 Key assumptions and limitations

1.5.1 Assumptions

One main issue when researching on the large amount of data that is available in this project is to make sure that the quality of the data is good. The challenge will be to make sure that the reference data set indicates for each data point if it is a failure-prone or non-failure-prone situation. For instance, if a server is shut down manually the monitoring data will show that the server status is down and incorrectly indicate a failure-prone situation. To make sure we have the best quality possible for this research we have selected data only from production servers, because they are very infrequently restarted or turned off.

1.5.2 Limitations

This project will be limited to search for patterns in UNIX metrics that can give an online prediction whether a server is failure-prone or not. This includes making UNIX shell

scripts to automate the process of preparing the data from the database into two category-files that will be used for classification. The online failure prediction would most certainly be more accurate if several methods where used in conjunction. Due to limited time we will not investigate any other methods This means that the research is limited to using generic operating system metrics like used memory, disk, i/o, etc, for symptom-based online failure prediction with classifiers.

1.6 Contributions

In this thesis we evaluate a novel approach to online failure predicting for UNIX operating systems using open source software. The evaluation is done with real world data from a performance monitoring system. We have used approximately thirteen month with performance data from sixteen failure-prone production servers for the evaluation. Our intention has been to make all parts of this solution fast enough to run online. For this reason we analyze all the features with Forward stepwise selection to speed up and get a more accurate prediction. For classifying we use Naive Bayes and Logistic Regression. Both of the classifiers are evaluated with features in a standard format and in a vectorized format in order to find differences in prediction speed and accuracy. We use a novel approach where generic UNIX performance metrics like memory use, CPU utilization and I/O traffic are used as input data for Naive Bayes and Logistic Regression to achieve failure prediction for software. We believe that the use of open source software and operating system commands that are generic for all UNIX genres to gather the features for online failure prediction gives a good advantage for further research in this area, because the software and code can be implemented relatively easy and without any extra expenses for the software. The research has determined which performance metrics are suitable for online failure prediction and used forward selection of features to make a generic set of the most significant features that can be used on any UNIX server for failure prediction. This helps to identify the most common symptoms for failure-prone situations. Used server memory is the most significant feature and thus it indicates that there is some memory leak in many of the servers investigated in this research. We have proved that the Naive Bayesian classifier also for this task gives the best predictions and computing speed. The research shows that Logistic Regression is a good alternative if one takes into consideration the nature of the classifier and eliminate all features with constant values before classifying. We are convinced that this research shows that it is possible to find patterns

in the stored performance data that can be used to give a warning if the current situation is failure prone and thus give a continuous measure that judges the current situation as more or less failure-prone, although there is still some work that has do be done to achieve this. Our research together with the questions that we uncover also give a good foundation for further research.

1.7 Target audience

The target audience of this thesis is anyone interested in data mining, machine learning and classifiers. The thesis requires that the reader is familiar with basic concepts from statistics and classification theory, however the thesis is written in such a way that it should be possible for any interested reader to follow. A brief introduction to the most important concepts and theory is given, but especially with regards to statistics and probability distributions we assume some background knowledge.

1.8 Report outline

We start in the next chapter with an outline possible methods for online failure prediction and give a brief introduction to the theoretical framework. Then we explain and discuss the selected research approach in chapter 3, We use our novel approach in chapter 4 to analyze the performance data. and discuss the results in Chapter 5, before we conclude and propose further work in chapter 6.

Chapter 2

Outline of failure prediction methods and theoretical framework

First we start giving an overview of the possible failure prediction methods for the theses and in the next sections we give a brief description of each main category. For each method and principle approach we give a brief introduction to the theoretical framework. The articles referenced in the introduction are mentioned in the appropriate sub chapter to give a better understanding of the content in each article and picture what area the research belongs to. Sub chapter 4.2 describes the selected, most feasible method for this research and the possible principle approaches.

2.1 Synopsis of prior research

Felix Salfner et. al. [10] made a survey of online failure prediction methods in 2010. From this survey we have made a figure to give an overview of the possible research methods. Online failure prediction methods can be divided into four main categories according to the type of input data that is processed. The main categories can be divided further by the principle of approach they employ and then subsequently by the methods they use. The dotted lines in figure 2 indicate where further research methods exist, but they are left out because the approaches are not feasible.

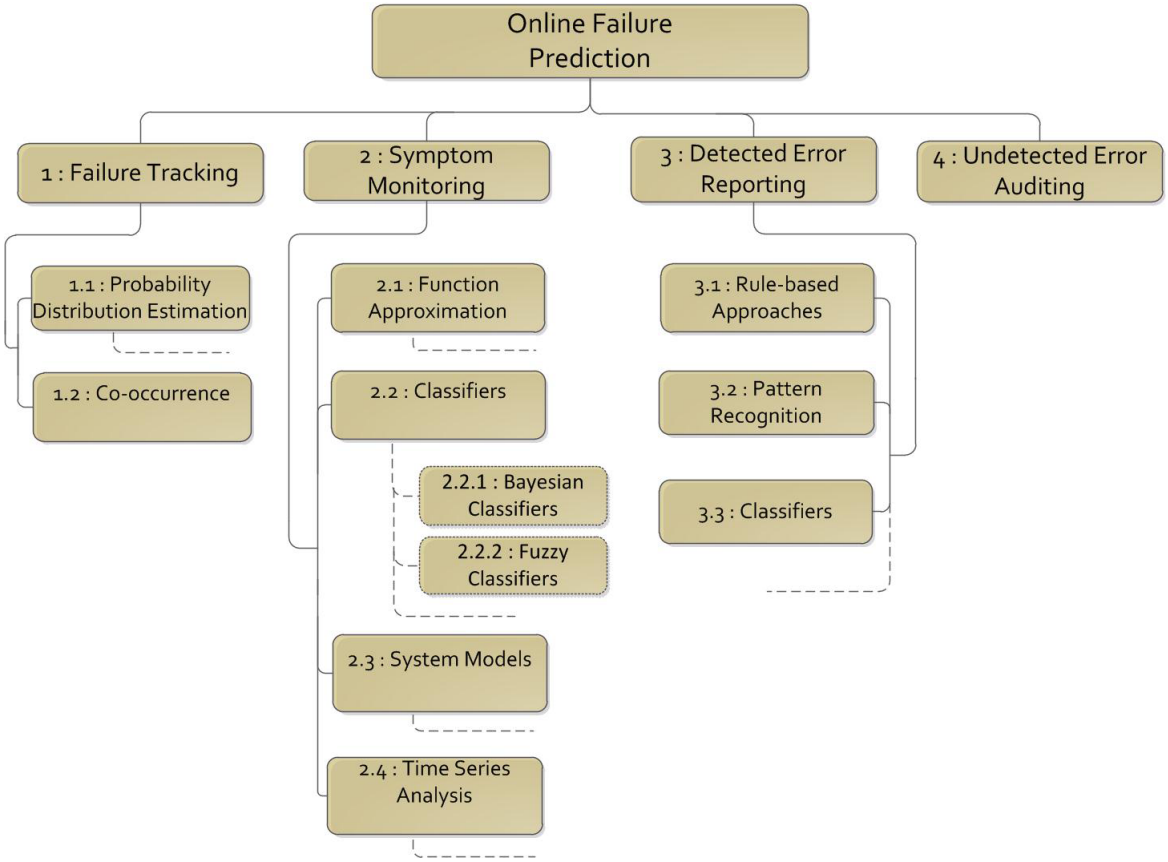


Figure 2.1: An overview over online failure prediction approaches. The dashed lines indicate further research methods. Composed from [10]

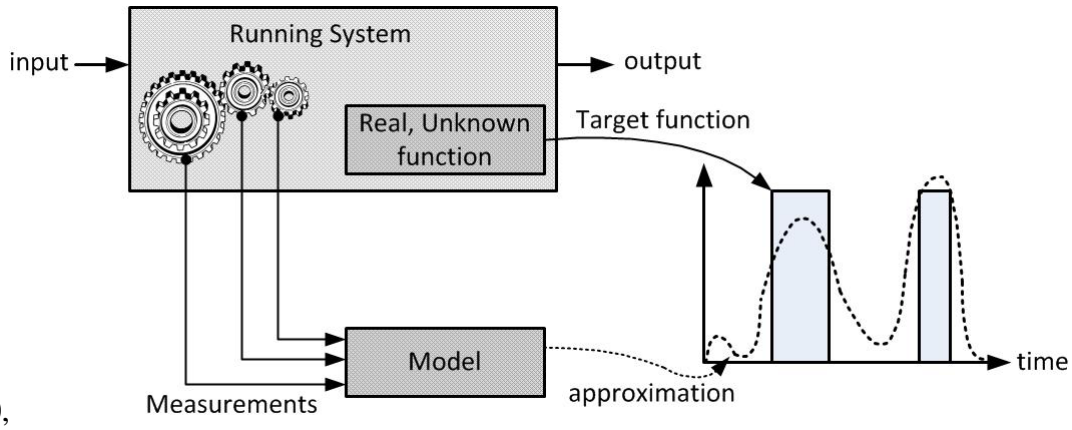
2.1.1 Failure Tracking

This method evaluates the times of occurrence as well as the types of failures that has previously occurred in a system. The failure prediction can be done in two different ways

1. Estimation of the probability distribution of a random variable for time to the next failure.

2. Approaches that build on the co-occurrence of failure events. Liang et al. [21] in 2006 use such an approach to predict failures of IBM's BlueGene/L from event logs containing reliability, availability and serviceability data. This is briefly described in chapter 1.1.

These techniques assumes the system is stationary within some time window and in the cases of:



0,

Figure 2.2: Function approximation. Composed from [10]

- bug-fixing
- configuration changes
- or even varying utilization patterns.

it will affect the failure process and it can result in poor estimations. Because of the weakness in this method it has not been considered for this research.

2.1.2 Symptom monitoring

Symptom monitoring assumes that symptoms are side-effects of errors. Thus the approaches evaluate monitoring data reflecting symptoms (side-effects) of errors. The intention of failure prediction based on monitoring data is that errors like memory leaks can be detected by their side effects on the system, such as unusual memory usage, CPU load, disk I/O, or unusual function calls in the system. These side effects are called symptoms. Symptom-based online failure prediction methods frequently address non-fail-stop failures, which are usually more difficult to grasp. This method is the most adequate method for the monitored data available for this project. To find patterns in the UNIX performance metrics is analogous to analyzing monitoring data in order to detect symptoms that indicate an upcoming failure. There are four principle approaches that have been identified: Failure prediction based on:

2.1.2.1 Function approximation

Function approximation tries to imitate an unknown target function by the use of measurements taken from a system at run-time (see fig. 2.2). For failure prediction the target function is usually the probability of failure occurrence, where the target value is a Boolean variable only available in the training data set but not during run-time, or it might be some computing resource such as the amount of free memory. Although the current value of free memory is measurable during run-time, function approximation is used in order to extrapolate resource usage into the future and to predict the time of resource exhaustion.

2.1.2.2 Classifiers

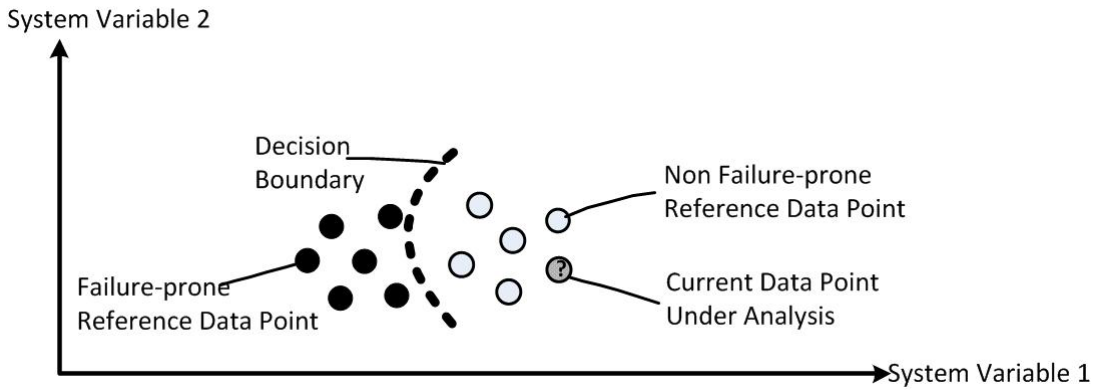


Figure 2.3: Online failure prediction by classification of operating system metrics. Composed from [10]

Instead of approximating a target function, some failure prediction algorithms evaluate the current values of system variables directly. Failure prediction is achieved by classifying whether the current situation is failure-prone or not. This is shown in figure 4.
The classifier's decision boundary is usually derived from a reference data set for which it is known for each data point whether it indicates a failure-prone or non-failure-prone situation. Online failure prediction during run-time is then accomplished by checking on which side of the decision boundary the current monitoring values are. The dimensions of data points can be discrete or continuous values. We have briefly described the research of Hamerly and Elkan [12] where they use the Naive Bayes classifier for hardware failure prediction.

Classification methods evaluated in this thesis are:

K-nearest neighbor[23] K-nearest-neighbor classification is one of the most fundamental and simple classification methods and should be one of the first choices for a classification study when there is little or no prior knowledge about the distribution of the data. K-nearest-neighbor has the advantage that it is robust to noisy training data (especially if we use inverse square of weighted distance as the "distance") and it is effective if the training data is large. K-nearest-neighbor has disadvantage that it needs to determine the value of parameter K (number of nearest neighbors) and for the distance based learning is not clear which type of distance to use and which attribute to use to produce the best results. Shall we use all attributes or certain attributes only? Also the computation cost is quite high because we need to compute distance of each query instance to all training samples. There exist some indexing methods that may reduce this computational cost.

Bayesian classifiers [7] A Bayes classifier is a simple probabilistic classifier based on applying Bayes theorem (from Bayesian statistics) with strong (naive) independence assumptions. In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 4" in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple. Depending on the precise nature of the probability model, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without believing in Bayesian probability or using any Bayesian methods. An advantage of the naive Bayes classifier is that it requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix.

Logistic Regression [3, 22] Logistic regression is a popular classification method that comes from statistics.and is used for prediction of the probability of occurrence of an

event by fitting data to a logit function. This is a useful way of describing the relationship between independent variables (in our case e.g. used server memory, disk i/o, etc.) and a binary response variable, expressed as a probability, that has only two values, such as being failure-prone or not ("failure-prone behavior" or "normal behavior"). It is a generalized linear model used for binomial regression. Like many forms of regression analysis, it makes use of several predictor variables that may be either numerical or categorical. The logistic function is useful because it can take as an input any value from negative infinity to positive infinity, whereas the output (probability (p)) is confined to values between 0 and 1. The probability (p) of a class value is computed as:

p = f(F) = exp(F)/(1+exp(F))

The variable F represents the exposure to some set of independent variables, while f(F) represents the probability of a particular outcome, given that set of explanatory variables. For our classifier in Orange the outcome variable F (class) must be binary (dichotomous) and discrete attributes must be translated to continuous. The variable F is a measure of the total contribution of all the independent variables used in the model and is known as the logit. The model is described by a linear combination of coefficients, where the variable F is defined as:

f(F) = ß_0 + ß_1*X_1 + ß_2*X_2 + ... + ß_k*X_k

where ß0 is called the "intercept" and ß_1, ß_2, ß_3, and so on, are called the "regression coefficients" of X_1, X_2, X_3 respectively. The intercept is the value of z when the value of all independent variables are zero (e.g. the value of z in normal server behavior). Each of the regression coefficients describes the size of the contribution of that risk factor. A positive regression coefficient means that the explanatory variable increases the probability of the outcome, while a negative regression coefficient means that the variable decreases the probability of that outcome; a large regression coefficient means that the risk factor strongly influences the probability of that outcome, while a near-zero regression coefficient means that that risk factor has little influence on the probability of that outcome.

Logistic regression tends to systematically overestimate odds ratios or beta coefficients when the sample size is less than about 500. With increasing sample size, the magnitude of overestimation diminishes and the estimated odds ratio asymptotically approaches the true population value. In a single study, overestimation due to small sample size might not have any relevance for the interpretation of the results, since it is much lower than the standard error of the estimate. However, if a number of small studies with system-

atically overestimated effects are pooled together without consideration of this effect, an effect may be perceived when in reality it does not exist.[22] A minimum of 10 events per independent variable has been recommended.[3] For example, in a study where death is the outcome of interest, and 50 of 100 patients die, the maximum number of independent variables the model can support is 50/10 = 5.

2.1.2.3 A system model

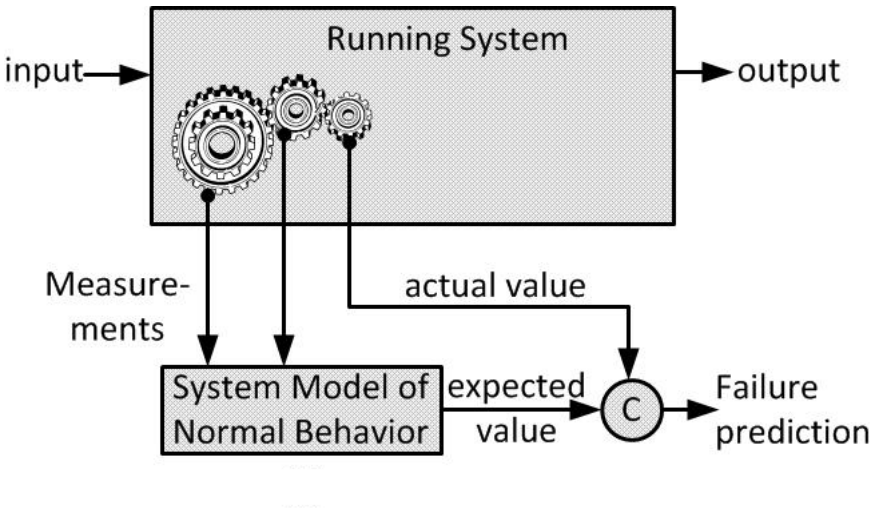


Figure 2.4: Online failure prediction using a system model. Composed from [10]

In contrast to the classifier approach, which requires training data for both the failure-prone and non-failure-prone case, system model based failure prediction approaches rely on modeling of failure-free behavior only (normal system behavior). The model is used to compute expected values, to which the current measured values are compared. If they differ significantly, the system is suspected not to behave as normal and an upcoming failure is predicted. We have available training data for both the failure-prone and non-failure-prone case and therefore do not consider this method.

2.1.2.4 Time series analysis

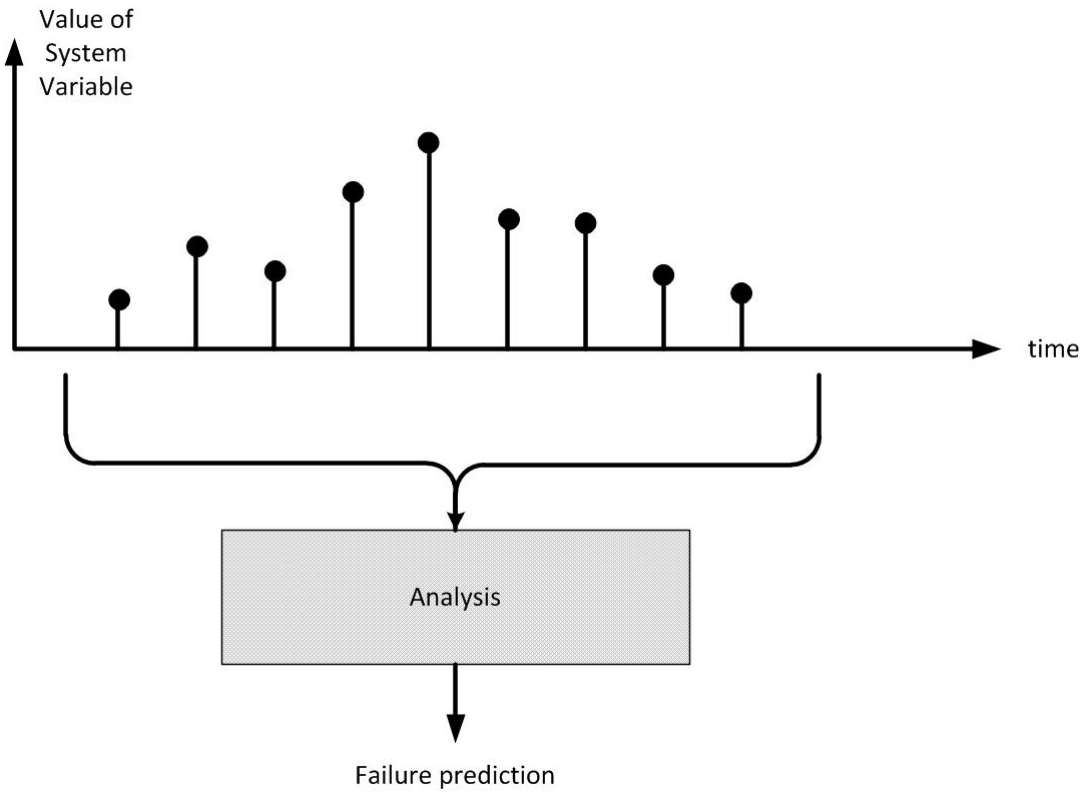


Figure 2.5: Online failure prediction using time series analysis. Composed from [10]

As the name suggests, failure prediction approaches in this category, treat a sequence of monitored system variables as a time series. This means that the prediction is based on an analysis of several successive samples of a system variable (see figure 6). The analysis of the time series either involves computation of a residual value on which the current situation is judged to be failure-prone or not, or the future progression of the time series is predicted in order to estimate, for example, time until resource exhaustion. This approach does not seem to give results as good as with classifiers and we have not considered it because of this.. In chapter 1.1 we briefly describe two references using time series analysis. Castelli et al. [5] in 2001 say that IBM has implemented a curve fitting algorithm for the xSeries Software Rejuvenation Agent and Cheng et al. [6] in 2005 presents a two step approach for failure prediction in a high availability cluster system.

2.1.3 Detected error reporting

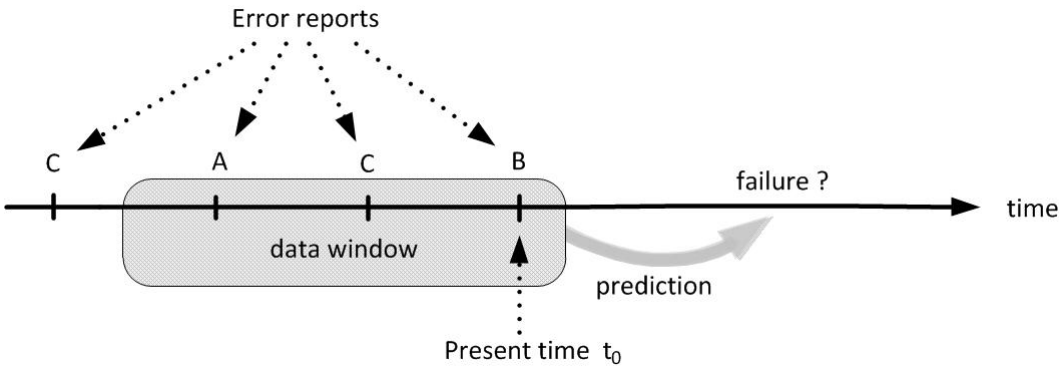


Figure 2.6: Online failure prediction based on error reports. Composed from [10]

Analyze error reports to detect errors that have not yet evolved to become a failure. A log file is a list of messages and an error log provides a mechanism for reporting errors, warnings, and other significant events that happen during run-time of a server. Each message written to the error log will include a category (indicating the area of the server in which the message was generated) and severity (indicating the relative importance of the message), along with an integer value that uniquely identifies the associated message string. The error log can be analyzed for:

- Auditing, determine the cause of an event (past).
- Predicting important events (future).

The different approaches can be divided in two main groups:

1. Rule-based failure prediction methods derive a set of rules where each rule consists of error reports.
2. Error pattern-based approaches has several techniques:

   (a) Co-occurrence of errors is a method used by many researchers with good results. Salfner and Malek in 2007 [26] use error log files with data from a commercial telecommunication system as input data. Lal and Choi in 1998 [24] used log files from a UNIX server, Leangsuksun et al. in 2004 [19] presented a study where hardware sensors measurements such as fan speed, temperature, etc are used as input. The method is briefly described in chapter 1.1.

(b) Pattern recognition techniques that operate on sequences of error events trying to identify patterns that indicate a failure-prone system state. We briefly describe the research of Salfner and Malek [26] in chapter 1.1 that uses error log files with data from a commercial telecommunication system as input data to predict performance failures.

(c) Statistical test

(d) Classifiers

Detected error reporting is a method that is proven to give good results in above mentioned researches and it would be very interesting to evaluate on its own or in conjunction with the "symptom monitoring", but we choose not to use it because of the limited time for the thesis.

2.1.4 Undetected error auditing

Using auditing to actively search for incorrect states (undetected errors) regardless whether the data is used at the moment or not. Undetected error auditing uses auditing to actively search for incorrect states (undetected errors) regardless whether the data is used at the moment or not is a method that has not been researched yet [10]. One drawback with this method is that it can cause a high load because it is online. Experience with security auditing has shown that this can be a serious problem. In a production environment this is not realistic to implement.

Chapter 3

Research approach

I start this chapter I will first explain how the research question has emerged. In section 3.1 we select the research approach and in section 3.2 we state the requirements before we in section 3.3 explain how we select data to predict failures. The next two sections describes how the research is conducted to achieve online failure prediction. In the last section in this chapter we describe how to verify the achieved results.

In SITS we have an online performance monitoring system that shows any current performance issues for approximately 250 AIX/Linux servers running different Oracle products. The main (start homepage) page in the PMS shows the servers that currently have most important issues, for instance server down. The PMS is used by operation managers on duty to monitor the health condition of approximately 250 servers with UNIX operating system. The PMS system gathers performance metrics from all the servers every five minute with UNIX shell commands and stores 28 different performance metrics, for each

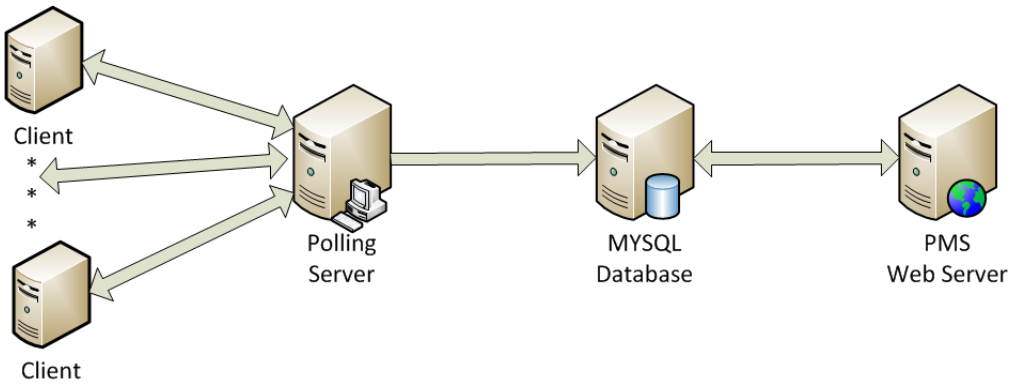


Figure 3.1: Model of the Performance Monitoring System

server in a MYSQL database. The performance metrics are stored for 12 months and the historical data can be viewed as graphs in order to analyze the server behavior, e.g to find out when a server failure occurred the first time and what can have caused the failure. This information is used to generate a web page that shows the servers that currently have most important maintenance issues, for instance if a server is down or percentage of filling for a file system. The servers with maintenance issues are listed by priority, where production servers that have failed are the most important issues. The page is regularly checked manually and some action will be taken if the person on call (watch) finds it necessary.
SITS wishes to improve the performance monitoring system, in order to improve the availability for all the systems that are monitored. The performance monitoring system should if possible give warnings or list the most failure-prone servers when it is probable that a system will fail in the near future.
The problem to address in this thesis is to identify "during run-time" whether a failure will occur in the near future based on an assessment of the monitored current UNIX system state.

This type of failure prediction is called "online failure prediction" and thus we have the research question: ”Online failure prediction in UNIX systems”
In this context we assume that a failure will cause a server and/or service to stop working, In the next sections we describe how the task is to be solved. Where the task is to make it possible to achieve reliable online failure prediction with the help of UNIX operating system metrics in order to identify situations that most probably will evolve into a failure. In the next Section we argues for the choice of method and in Section 3.1 we propose a model.

3.1 Selecting research method

To answer the research question above we must look for patterns in the database with monitored UNIX operating system performance metrics that can give a prediction of a possible imminent failure. The outline of failure prediction methods in the previous chapter revealed four possible failure prediction methods based on the type of input data they process. For each of the main methods we explain why or why not it is selected as a research approach:

Failure tracking techniques assumes the system is stationary within a time window and in

the cases of: bug-fixing, configuration changes or even varying utilization patterns it will affect the failure process and it can result in poor estimations. Because of the weakness in this method it has not been considered for this research.

Symptom monitoring assumes that symptoms are side-effects of errors. Thus the approaches evaluate monitoring data reflecting symptoms (side-effects) of errors. Symptom monitoring is a method that analyzes the monitored data in order to detect symptoms that indicate an upcoming failure. Monitored metrics from the Performance Monitoring System is available for this thesis and symptoms monitoring with classifiers is the optimal method to use on the available performance data. Thus we select symptoms monitoring with classifiers to be used as the research approach for this thesis.

Detected error reporting is a method that analyzes the error reports on a system to detect errors that have not yet evolved to become a failure. This method could also be used for this thesis, alone or in conjunction with symptom monitoring. There are several reasons why we have choosen not to use this method. From state of the art research we have:

- Felix Salfner and Steffen Tschirpke [11] in the paper "Error Log Processing for Accurate Failure Prediction" from 2008, conclude that even though error logs are a fruitful source of information both for analysis after failure and for proactive fault handling. However, in order to get access to the essential information contained in error logs, the error log data needs to be filtered and formatted into shape, so that valuable pieces of information can be picked from the vast amount of data stored in error logs. The authors state "The results unveiled that elaborate data preparation is a very important step to achieve good prediction accuracy." From the conclusion we can tell that it is a very time-consuming process to achieve good results for online failure predictions when using error logs.

- One of the major findings in Hoffmann et al.[2007][14] was that the issue of choosing a good subset of input variables has a much greater influence on prediction accuracy than the choice of modeling method. This means that the result will improve when concentrating on choosing the input variables.
  From the above we have that the use of error logs for online failure predicting is a very time demanding task and that focusing on the selection of the input variables will give better results compared to the time spent.

Detected error reporting is a method that is proven to give good result in above mentioned researches and it would be very interesting to evaluate it alone or in conjunction with the "symptom monitoring", but we choose not to use it because of the limited time for the thesis. The error log files for the servers used in this thesis are not configured in a standardized manner and it would not be possible to change the logging strategy, set up filtering and selecting the valued information for all the 250 servers within the amount of time available for this thesis The method "detected error reporting" will for these reasons not be used for this theses.

Undetected error auditing use auditing to actively search for incorrect states (undetected errors) regardless whether the data is used at the moment or not. One drawback with this method is that it can cause a high load because it is online and has to log from the most active processes. Our own experiences with security auditing has shown that this often is a serious problem to the pay-load of a server and we do not consider this to be a useful approach.

In the next section we investigate the research question “Online failure prediction” with respect to the area of application

3.2 Selecting a model

Using symptom monitoring to get online failure prediction we must automate the process for predicting how failure prone a server is at the present time. The following algorithm gives a brief introduction of what has to be done to set up an environment for predicting online failures:

Online failure prediction algorithm for one server:

1. Select performance data for a specific host from the PMS MYSQL database and save to tabulator separated text file host-name_date.tab.
2. With the help of a Unix shell script prepare a data-set with two classes from the file host-name_date.tab.
   (a) One class that represents normal server behavior (drawing a baseline)

   (b) and the other class which represents the behavior a preset time period prior to server failure (critical). This data-set is saved to a file host-name_date_data set.tab.

3. Use a Python script with Orange modules to generate predictions of the failure prone state of the server.

4. Save predictions in the PMS MYSQL database so the information can be used to show a page with status for the most failure prone servers and give a warning by e-mail or SMS.

With the online failure prediction algorithm in place several questions emerge:

- Which of the UNIX performance metrics should we use for failure prediction?
- Will virtual servers show the same result as for physical servers, given the same operating system metrics?
- Which combinations of monitored metrics gives the best result for predicting online failures?
- Which classifier gives the best predictions, for instance the Bayesian classifier?
- Which of the classifiers gives the best performance in terms of:
  - Speed.
  - Accuracy.
  - Speed compared to accuracy.
- How early can the failure prone pattern be found?
- Is possible to give a binary decision or give a continuous measure that judges the current situation as more or less failure-prone?
- Can it be done for many computers simultaneous, how many?

Each question is treated as a separate experiment and they are described separately in the following paragraphs:

Which of the UNIX performance metrics are suitable for failure prediction? When investigating the above question to find reasons for different behavior, we ask our selves: Why may the OS metrics that have significant values in predicting online failures change for different computers?

- The computers may have different hardware e.g. Motherboard, CPU etc.. Some hardware is more failure prone than other.
- The installed software is different for most computers, except for cloned computers. A cloned computer, is a computer installed with an exact copy of the software from a reference computer or it can be a virtual computer which is an exact copy of another virtual computer and the only things that differ is host name, media access card address etc..Software is failure prone because of bugs, memory leaks etc.
- Computer load: High load on a computer may result in trashing and this can cause services to stop responding.
- Find all the metrics that does not have any impact for online prediction. This might be monitored metrics that have more or less constant values.

First we examine each fields in the PMS data to find all the features that reflect the change of system performance and fluctuate according to systems change. Static fields does not reflect the system performance and can be eliminated. When we have all the features (performance metrics) that reflect the systems change we want to find....

Which combinations of monitored metrics gives the best result for predicting online failures? In order to find the optimal set of features we have to predict with different combinations of the features and measure each combination. We mentioned in section 1.1 that feature selection has proved to be very important in several researches e.g. Hoffmann et al. [2007][14] and that a typical variable selection algorithms is Forward Stepwise Selection (see, e.g., Hastie et al. [13] , (Chapter 3.4.1). The method has been used by Turnbull and Alldrin [28] and Hoffmann [2] with good result. With relatively few features to evaluate (max 27) this should not be to costly to try out. To find a set of features that are generic for all the servers we will calculate the intermediate values in each step of Forward stepwise selection process and pick the most significant intermediate value for the next step. This test should give a list of the metrics with their significance to prediction accuracy. Once we have

established which metrics that can add value for the prediction of online failures we still do not know if:

virtual servers show the same result as for physical servers, given the same operating system metrics. First we need to define a virtual server: In this thesis hardware virtualization or platform virtualization defines a virtual server that acts like a real computer with an operating system. The software executed on these virtual servers are separated from the underlying hardware resources. The term host computer refers to the actual computer on which the virtualization takes place; the term guest computer, refers to the virtual server. The virtual servers investigated in this thesis are the following:

- Xen virtual servers for Red Hat Linux running on Hewlett Packard intel host servers
- AIX Power 570/795 host servers running AIX 6.0 with IBM PowerVM virtual servers

The use of virtualization has many benefits for instance:

- Increase the hardware utilization.
- Reduces physical space and operating costs involved with powering and cooling older, often less efficient computers.
- Decrease the capital and operating cost by sharing in number of VM's.
- High availability and it is secure.

But we would think that a VM has a higher failure rate because of the extra layers with software or is it developed in such a manner that it compensates for all the "extra" program code? Counting the number of times virtual servers are down and comparing with the number times physical servers are down on the same amount of time should indicate if there is a difference or running separate test on physical and virtual servers to find which type of operating system parameters gives the best prediction result and which type of server gives the most accurate online failure predictions. Now that we know we will use as input for prediction, we have to find out:

Which type of classifier gives the best predictions. To find the answer to this question we simply try to predict with different classifiers e.g.:

- KNN classifier
- Bayesian classifier
- Logistic Regression etc.

For each classifier evaluated we want to find as many of their properties as possible, such as:

- Speed: We want to find if there is a big difference in the time it takes to find a online failure prediction. Performance metrics are gathered approximately every five minute for each server. This means we want to do the prediction in less than the five minutes time window when the next set of performance metrics are gathered.
- Accuracy: A theoretical, optimal failure prediction aims to achieve 100% sensitivity, but how many valid failure predictions can be done if we want to be for instance 95 % sure that there is a upcoming failure. This can be measured with a ROC curve (Receiver Operating Characteristic). The ROC curve was first developed by electrical engineers and radar engineers during World War II for detecting enemy objects with radar, and was known as the signal detection theory Since that time ROC analysis has been used in many areas like medicine, radiology, and lately it has been introduced in machine learning and data mining.[8, 9] The area closest to the north-west of the ROC curve, shows the best performance of the tested classifier.
- Speed compared to accuracy: For use of the online failure prediction in the PMS we need make sure that there are very few false positives. this is due to earlier basis of experience, where the threshold systems have turned out to be useless because of to many false positives. If we set a threshold limit for the failure prediction to be very high, for instance it should be 99 % sure that there is an upcoming failure in the near future, will this affect the time it takes to predict a upcoming failure?
- Computing load: How many predicting processes can a server handle? Is it possible to do online failure prediction for all the servers in PMS (approximately 250 servers)? Is it possible to do failure predictions with the data power from a single computer for all the servers in PMS simultaneously? If not, how many computers are needed for this task. In the worst case scenario we might have to do the failure prediction's on each host being monitored by PMS.

We should now be able to predict failures, but how early can the failure prone pattern be found? Is it possible to predict an upcoming failure minutes, hours, days or even weeks before the failure comes into force? If time permits we will investigate if an upcoming failure can be predicted relatively long before the actual failure. This can be done by changing the size of the classes used for failure predicting. The script that makes the data sets must have a variable that declares a time interval before a failure occurs. In this way we can look for failure-prone patterns as far in advance as possible.. Here we need to take care of challenges like:

- The time between two failures: what if two or more failures occur inside a time period shorter than the one we have set for the failure-prone class? To avoid this we need to check, and find the smallest time-window between to failures and set the initial time-window to this value and then try smaller time-windows for each step.
- What is the minimum amount of failures we need to train the classifier to achieve a acceptable sensitivity for a failure prediction. If we are able to determine this we can set a threshold in order to not give inaccurate values if the classifier is not properly trained.

When the failure prediction model is established, it is essential to analyze the quality to find out if it can:

Give a binary decision or a continuous measure that judges the current situation as more or less failure-prone. If a known error that causes a failure can be predicted, and we know that the failure prediction accuracy is above a given threshold, then we can give a binary decision to schedule a counter measure, for instance restarting some software that has a memory leak or other known error that will result in a failure. If the failure prediction accuracy is below a given threshold, then we can use the prediction to give a continuous measure that judges the current situation as more or less failure-prone. Storing the measure in the PMS database can be used to give a list with the most failure prone server on top. The history can give a graph which will be useful for testing quality of upgrades or new software.

Finally we will make sure our requirements are met and to be careful and validate the research by comparing two different classifiers when solving the research problem. Enabling cross-validating techniques should be done if possible. In this way we avoid failures in solving the problem under investigation.

Chapter 4

A novel approach to failure predicting

This chapter explains how the research is conducted to achieve online failure prediction. In section 4.1 we state the requirements before we in section 4.2 introduce the model. Section 4.3 explains how we select data to predict failures and show how the data is preprocessed. Feature selection is introduced in section 4.4 and in section 4.5 we evaluate the model. The last section of this chapter explains how the achieved results are verified.

4.1 Requirements

To make sure the Performance Monitoring System in SITS benefits as much as possible from this research we strive to fulfill the following requirements:

1. Make it portable to any UNIX operating system.
2. Use open source software so that the research methods can be performed free of cost. Software used is:
    - Unix shell script
    - MYSQL[27] open source database used for the PMS.
    - JpGraph[15] JpGraph is an Object-Oriented Graph creating library for PHP >= 5.1 and is released under a dual license. QPL 1.0 (Qt Free License) For non-commercial, open-source or educational use. Also used for the PMS.
    - Python[16] open source programming language
    - Orange[18] Open source data mining through visual programming or Python scripting. (import Orange library into Python)

4.2 Failure prediction model

Figure 4.1 is a offline model which is used to train the online failure prediction model depicted in figure 4.2.

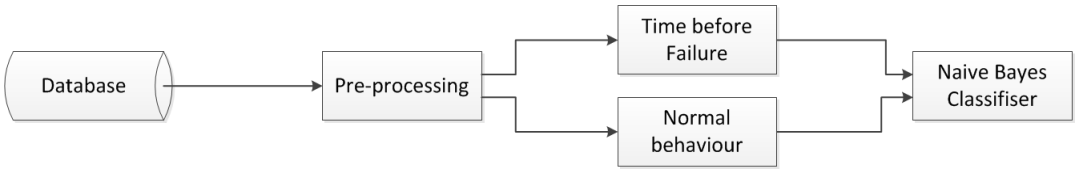


Figure 4.1: Offline model for the training process

With the requirement in place we start by drawing a training model for failure prediction with classifiers. The raw performance data is extracted from the database and fed into a preprocessing utility in order to make the data into a understandable (supervised) format for the classifier. Naive Bayes classification requires that input variables take on discrete values. Therefore, the monitoring values are assigned to a discrete class in the preprocessing utility. We define a "critical" class that holds a time window of monitoring data, that starts from a period in time before a recorded failure and ends with the time of failure. All the rest of the performance data is assigned to the class "baseline", which hold the data for normal operation. The preprocessed performance data is then passed on to the classifier for evaluation.

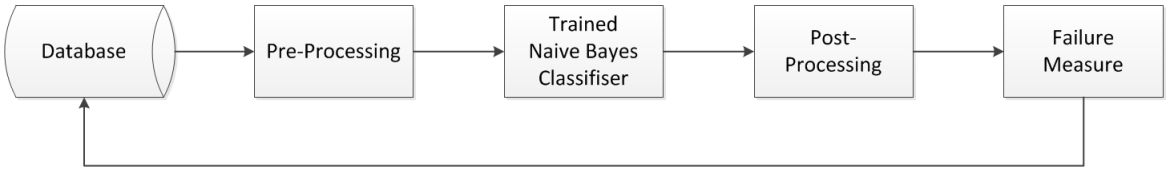


Figure 4.2: Online model for predicting of failures from symptoms

When the classifier is trained it is ready to classify online performance data. The prediction from the classifier is now post-processed in order to give a continoues measure or binary decision whether it indicates a failure-prone or non failure-prone situation. The failure measure is then stored in the database for later use, for instance show historical graph for root cause analyze.

4.3 Data Preprocessing

4.3.1 Selecting data to predict failures

In the first step of the process for predicting online failures we extract all the performance data. The monitored performance metrics from the different UNIX servers are stored in a MYSQL database. Every month a new table space is created to keep the size of the table space easier to handle with respect to backup restore.
The extraction is done with a SQL command. Selecting data for the specified host "xsru136in":

```
SELECT * FROM '201009_server-data'
WHERE node = "xsru136in"
INTO OUT-FILE 'c:/201009_host-name.tab'
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n';
```

The above command will extract all the records for one host(server) for the month the table-space yield and put the data in a tabulator separated file (date_host-name.tab).
In this thesis we will use data only from selected failure-prone production servers, because we know that these computers have recorded failures and are not turned off or restarted in office hours. They are only on rare occasions stopped or restarted manually after maintenance e.g. a software upgrade. All 16 servers used in the research are selected from a manual registration system for "down time", which listed them as failure-prone.

A record from the PMS database with labels for a specific host has the following format:

```
id sgroup uarea dato node up oslevel sload CPU cputot entcpu lcpu smem
    memtot fcache swap pageing iowait network oradb current errlist tps wq
    ec pc sr fr netio
```

A record from an intranet server has the values :

```
3198283 SKATTENETT REF_DB 20100901000135 host name UP 5300-09
    1,18 97 4 1,00 8 80 3072 213 12 0 0 179 2 0 0 2 0 3,8 0,04 0 0 60
```

See the table 4.1 below for a explanation of each field in a record

Field	Description of the fields
id	Primary key in the PMS database.
sgroup	The servers are grouped into systems serving a purpose e.g.. Intranet, CITRIX etc.
uarea	Used in area: Development, test, reference or production.
dato	date and time when the record is saved.
node	Server host name.
up	If the host is not UP it may be DOWN or have the status ERR.
oslevel	Operating system version and level.
sload	Server load in percent.
CPU	Percent of CPU power used.
cputot	Total available CPU's for the host system.
entcpu	Entitled processor capacity used in IBM's Power systems [17].Commitment of capacity that is reserved for the partition. Set upper limit of processor utilization for capped partitions.
lcpu	Logical CPUs are the number of virtual CPU's available for the host.
smem	Server memory used.
memtot	Total available server memory.
fcache	Memory used for file-cache.
swap	Memory swapped to file.
pageing	Is a memory-management scheme with fixed-size blocks called pages by which a computer can store and retrieve data from disk for use in main memory to allow the physical address space of a process to be noncontinuous.
iowait	Time waited for in/out services.
network	Average turn - return time when running the shell PING command.
oradb	How many Oracle databases are running on this host.
current	Not relevant.
errlist	Hardware failures.
tps	Indicates the number of transfers per second that were issued to the physical disk. The second report generated by the iostat command is the disk/tape utilization report. The disk report provides statistics on a per-physical-disk basis. A transfer is an I/O request to the physical disk/tape.
wq	Waitqueue: Is a vmstat sub command (column b:). The average number of threads that were waiting for paging to complete. Average number of kernel threads placed in the (VMM) wait queue (awaiting resource, awaiting input/output) over the sampling interval. CPU IO wait %.
ec	The percentage of entitled CPU capacity consumed. A vmstat sub command specific for AIX. The vmstat command displays virtual memory statistics. This metric will only be displayed on a virtual environment if the partition is running with shared processor.
pc	Sub command from vmstat specific for AIX. The number of physical processors consumed.A vmstat sub command specific for AIX. The vmstat command displays virtual memory statistics.
sr	Pages scanned by page-replacement algorithm.
fr	Pages freed (page replacement).
netio	Network in/out traffic (mb/s).

Table 4.1: Available system metrics

In the next subsection we show how the data containing the performance metrics is prepared for training.

4.3.2 Preparing training data

Once the performance data for one host is exported from the database the next step is to format the records so they only have fields of interest for pattern recognition. This means extracting fields that can influence the learners and leave out fields that are static like id, Sgroup, Uarea, host-name etc.. We start with all the fields that possibly can play a role in determining a failure prone state:

- sload CPU lcpu smem memtot fcach swap pageing iowaait network oradb current errlist tps wq ec pc srfr netio

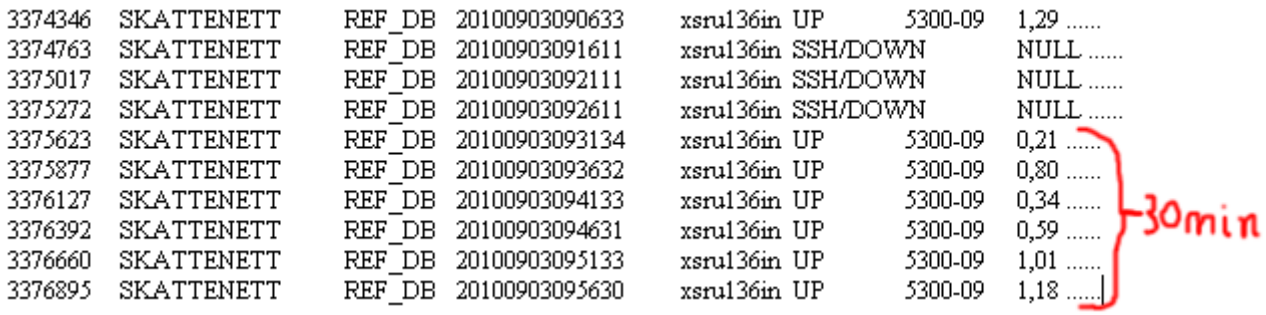


Figure 4.3: Example data from the PMS database

In order to recognize a pattern for a failure prone state in this data we need to look at a time period before a recorded failure. A failure can be found in the records which have the value of the field “UP”different from “UP”, for instance the value may be “SSH/down” or have the status “ERR”. Since we do not now the size of this time window where the failure prone pattern is most visible, we initially try with a 30 minutes time period before a failure.

Each record in the data is taken approximately every five minutes, so 30 minutes is six records from the PMS database. The file now holds the performance metrics of interest and we can now prepare the data for pattern recognition. To automate this operation we use a UNIX shell script. The algorithm shown beneath uses 30 minutes as an example for the time window before the failure occur.

An algorithm for preparing input data:

- Remove the fields that is of no interest from all records
- Reverse the order of the records so the oldest record comes first
- Look for records that have a value in column 2 that is different from "UP" and add the next 6 records (approximately a 30 minutes time period) to the file critical_records.tab and mark them with the class name critical
- Remove the two first fields from all records (Time stamp and status) which are no longer needed.
- Put all the records except the critical_records in the file baseline.tab and mark them with the class name baseline
- Open a new host name_prepared_data.tab and add column labels.
- Add critical_records.tab and then baseline.tab to the file host name_prepared_data.tab.
- Convert host name_prepared_data.tab from UNIX to DOS text format

This shell script shown as an algorithm, also has functionality to make a vector representation of the time window before the failure. The vectorization of the prepared data gives better results and performance for the pattern recognition in most cases. In the next subsection we explain features and how we select them.

4.4 Feature selection

Feature selection is the technique of selecting a subset of relevant features for building robust learning models, by removing the most irrelevant and redundant features from the data.
In order to find which operating system metrics combination that gives the best results we will try with different combinations of the metrics in separate trials. Each trial will eliminate one or more of the 16 metrics(features) that we start off with, until we are left only with the UNIX performance metrics affecting the performance of the learning model we use.

Feature definition: A sample's features are its individual measurable properties and characteristics [25]

With 16 features we have 15+14+...3+2+1=120 possible combinations that we have to try if we are to find the best combination for the performance data feature set. The dimensionality of the PMS data does not pose a challenge to the learning tasks due to the curse of dimensionality. But still with the existence of many irrelevant features, the learning model can over fit and become less comprehensible. To avoid this we use feature selection to identify relevant features for dimensionality reduction. Feature selection algorithms designed with different strategies mainly fall into three categories: wrapper, filter and embedded models.

Filters: Filter models rely on the general characteristics of data and evaluates features without involving any classifier. Filters use a search algorithm to search through the space of possible features and evaluates with a filter.

Wrappers: These models require a predetermined classifier and uses its performance as evaluation criteria to select features. Wrappers use a search algorithm to search through the space of possible features and evaluate each subset by running a model on the subset. Wrappers are computationally expensive and can over fit the model.

Embedded models: Embedded techniques are embedded in and specific to a model. Thus the algorithms with embedded model, incorporate variable selection as a part of the training process, and feature relevance is then obtained analytically from the objective of the learning model.

The space of possible features are relatively low in this case and will not be to computationally expensive. We therefore select the wrapper search algorithm "Forward stepwise selection": This is a data-driven model building approach. In this approach, we add features to the model one at a time. At each step, each feature that is not already in the model is tested for inclusion in the model. The most significant of these features is added to the model. Thus we begin with a model including the feature that is most significant in the initial analysis, and continue adding features until none of the remaining features are "significant" when added to the model.

4.5 Evaluating the model

Which the data prepared, and a plan for how to find the optimal features, we begin looking for patterns with ORANGE in order to answer the questions in chapter 3.

4.5.1 ORANGE model

ORANGE is open source software for data visualization and analysis, which is integrated with a scripting language called Python. The data mining can be done through visual programming or Python scripting to automate the procedure. Figure 4.2 below shows the model we have used with the classifiers that perform best.

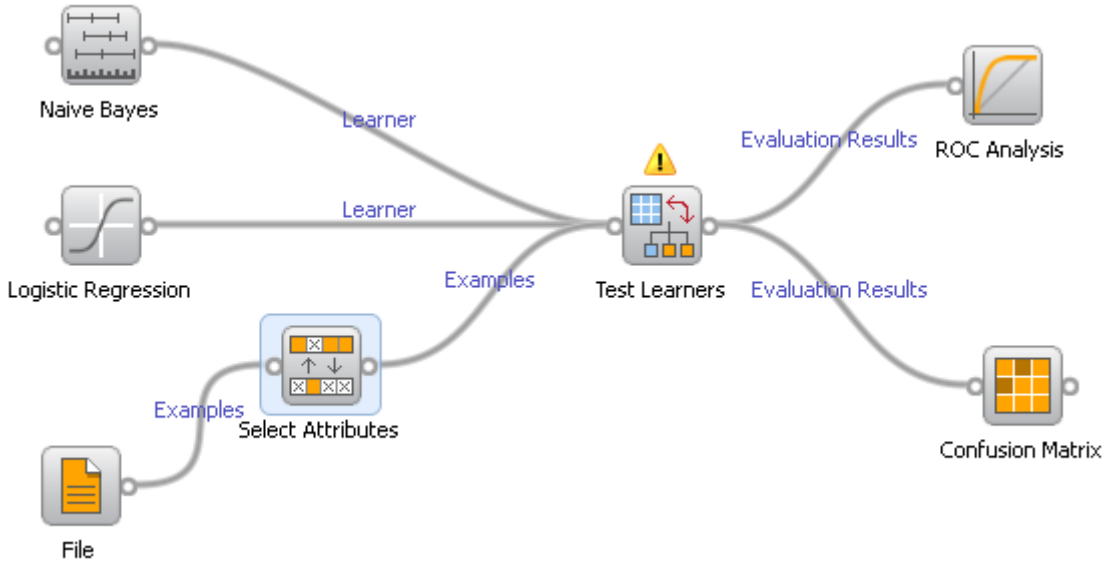


Figure 4.4: ORANGE Canvas with failure prediction model

The widget schema shown reads a PMS data set (File widget) and feeds the data to the Select attribute widget which is used to select all or any set of features. The idea is that some machine learning methods may perform better if they learn only from a selected subset of "best" features.The selected set of features is then fed to a Test Learner widget to evaluate the classifiers Naive Bayes and Logistic Regression on the PMS input data. The Test learner widget shows a table with different performance measures of the classifiers, such as classification accuracy and area under curve (ROC) and it outputs a signal with

data which is used by the ROC Analysis and Confusion matrix widgets to analyze the performance of the classifiers.

4.5.2 UNIX performance metrics suitable for failure prediction

The database with the PMS data has 29 fields in each record. Evaluating the lists of possible features in table 4.1 with respect to the definition that they are the "samples individual measurable properties and characteristics", we removed 15 possible features.

Id: The record primary *id* is obviously not a feature

SGROUP: The serving purpose of the server e.g.. Intranet, mail, etc. will of course affect the rate of failure, but since we are analyzing single servers and not the group this is not used as a feature.

uarea We only use servers from the production area and therefore do not need this as a feature.

dato The date field is used when preparing the data, but not as a feature.

node Is just the name of the server.

up The up field is used when preparing the data, but not as a feature.

oslevel Operating system version and level is used when preparing the data, but not as a feature.

cputot Total available CPU's for the host system is a static value and not used as a feature.

entcpu Entitled CPU's for the server is a static value and not used as a feature.

lcpu Logical CPUs for the server is a static value and not used as a feature.

memtot Total available memory for the server is a static value and not used as a feature.

oradb The number of Oracle databases that run on the server is a static value and not used as a feature.

current The current field is used by some part of the PMS script and is not a feature.

The suitable metrics for online failure prediction from the PMS data are all the fields that reflect the change of system performance and fluctuate according to systems change. There are 16 features that support this:

- sload CPU smem fcache swap pageing iowait network errlist tps wq ec pc sr fr netio

4.5.3 Optimal combination of features for predicting failures

Using the model in Figure 4.2 we do forward selection and start classifying. For each of the 16 servers we have classified the performance data using 10 fold cross validation for every feature in separate runs. Classifying is done with a standard formatted file with one set of features and with a vectorized data set, where a vector is 6 records, the size of the time window before a recorded failure. Pre-analyzing the classifiers we found that the best evaluation measure is the Area Under Curve (from ROC).

First step of forward selection is done by classifying with each feature in the data set in separate runs. Before we calculate the intermediate values we assessed alternative methods:

1. Calculate the intermediate values for all features without regard to what values they have. This gives the table below, which shows the intermediate values calculated from 16 servers with PMS data up to 13 month. The four columns to the left is a standard data set, and the four columns to the right are the vectorized data set.
   The classifiers are LR = Logistic Regression and NB = Naive Bayes.

Standard				Vectorized			
LR		NB		LR		NB	
Feature	AUC	Feature	AUC	Feature	AUC	Feature	AUC
errlist	0.5036	iowait	0.5346	pageing	0.5031	errlist	0.5018
pageing	0.5100	network	0.5348	errlist	0.5035	pageing	0.5031
iowait	0.5204	pageing	0.5361	fr	0.5215	iowait	0.5267
swap	0.5213	errlist	0.5410	swap	0.5220	wq	0.5267
network	0.5250	sr	0.5418	sr	0.5229	sr	0.5278
fr	0.5304	fr	0.5443	network	0.5231	swap	0.5295
sr	0.5318	swap	0.5480	iowait	0.5250	fr	0.5307
wq	0.5369	wq	0.5525	wq	0.5361	network	0.5453
sload	0.5442	CPU	0.5578	tps	0.5362	tps	0.5453
tps	0.5483	sload	0.5623	sload	0.5492	pc	0.5536
CPU	0.5612	pc	0.5662	ec	0.5549	CPU	0.5606
pc	0.5644	ec	0.5725	CPU	0.5569	sload	0.5637
ec	0.5666	tps	0.5744	pc	0.5593	ec	0.5730
netio	0.5752	netio	0.6116	netio	0.5767	netio	0.6011
fcache	0.6244	fcache	0.6273	fcach	0.6129	fcach	0.6077
smem	0.6370	smem	0.6381	smem	0.6237	smem	0.6077

Table 4.2: Results from first step of forward selection

As we can see from table 4.2 the standard data set got better AUC scores than the vectorized data set and the most significant feature is smem (used server memory) with AUC score equal to 0.6381. We also notice that the classifier that performs best is Naive Bayes.

2. Only use the AUC values that are larger than 0.5 for each feature, because values equal or below does not improve the classifier.

Standard		servers		Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
network	0.5819	swap	0.6246	network	0.6083	swap	0.6042
swap	0.5855	errlist	0.6273	CPU	0.6139	errlist	0.6076
errlist	0.5883	pageing	0.6276	tps	0.6179	fr	0.6109
pageing	0.5906	sr	0.6308	ec	0.6199	sr	0.6111
fcach	0.5937	fr	0.6310	pc	0.6200	network	0.6126
iowait	0.5962	network	0.6323	errlist	0.6268	wq	0.6160
tps	0.5977	iowait	0.6329	sload	0.6277	iowait	0.6188
netio	0.6007	CPU	0.6332	wq	0.6292	tps	0.6213
sload	0.6007	wq	0.6378	iowait	0.6315	pageing	0.6220
fr	0.6008	tps	0.6383	swap	0.6337	fcache	0.6246
sr	0.6008	pc	0.6435	pageing	0.6344	CPU	0.6315
ec	0.6067	fcach	0.6447	sr	0.6352	pc	0.6325
pc	0.6077	sload	0.6454	netio	0.6352	sload	0.6342
CPU	0.6116	ec	0.6557	fr	0.6360	ec	0.6429
wq	0.6118	netio	0.6720	fcache	0.6374	netio	0.6534

Table 4.3: Results from first step of forward selection, second alternative

3. Only use the AUC values that are equal or larger than the smem feature for each server in the past step, because we want to improve the classifier for each step in the process.

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
network	0.5819	swap	0.6246	errlist	0.5591	sr	0.5632
swap	0.5855	errlist	0.6273	pageing	0.5718	network	0.5671
errlist	0.5883	pageing	0.6276	network	0.6074	fr	0.5676
pageing	0.5906	sr	0.6308	CPU	0.6079	swap	0.5708
fcach	0.5937	fr	0.6310	ec	0.6188	errlist	0.5718
iowait	0.5962	network	0.6323	iowaait	0.6315	pageing	0.5802
tps	0.5977	iowait	0.6329	tps	0.6322	wq	0.5904
netio	0.6007	CPU	0.6332	wq	0.639	iowaait	0.6028
sload	0.6007	wq	0.6378	sload	0.6416	pc	0.604
fr	0.6008	tps	0.6383	sr	0.6425	tps	0.6121
sr	0.6008	pc	0.6435	pc	0.644	ec	0.6236
ec	0.6067	fcach	0.6447	netio	0.6474	CPU	0.6239
pc	0.6077	sload	0.6454	fcach	0.6537	sload	0.6376
CPU	0.6116	ec	0.6557	fr	0.6633	fcach	0.6463
wq	0.6118	netio	0.6720	swap	0.6747	netio	0.6582

Table 4.4: Results from first step of forward selection , third alternative

In table 4.3 the standard data set has the highest AUC scores, and the most significant feature is netio (network in/out traffic) with AUC score equal to 0.6534 when classified with Naive Bayes.

Looking at results from the three alternatives we find that netio has the largest AUC score in all cases. Evaluating the classifiers we also notice that some of the servers have feature values very different from the intermediate values, which indicates that the classifiers will gain from a feature selection process for each server. But if we are to find the most generic set of features for failure-prone UNIX servers the best alternative is to choose intermediate values for all features regardless of their value.

Second step in forward selection is done by classifying with the most significant feature smem from the previous step as a fixed feature, combined with all the features in the data set in separate runs.

Standard		servers		Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
network	0.5819	swap	0.6246	errlist	0.5689	swap	0.6042
swap	0.5855	errlist	0.6273	pageing	0.5763	network	0.6043
errlist	0.5883	pageing	0.6276	fr	0.5812	errlist	0.6076
pageing	0.5906	sr	0.6308	sr	0.5815	pageing	0.6091
fcach	0.5937	fr	0.6310	swap	0.5822	fr	0.6109
iowait	0.5962	network	0.6323	network	0.5851	sr	0.6111
tps	0.5977	iowait	0.6329	iowaait	0.5853	CPU	0.6127
netio	0.6007	CPU	0.6332	fcache	0.5861	sload	0.6153
sload	0.6007	wq	0.6378	netio	0.5868	wq	0.6160
fr	0.6008	tps	0.6383	tps	0.5913	fcache	0.6164
sr	0.6008	pc	0.6435	ec	0.5969	pc	0.6174
ec	0.6067	fcach	0.6447	wq	0.5990	iowaait	0.6188
pc	0.6077	sload	0.6454	sload	0.5992	tps	0.6213
CPU	0.6116	ec	0.6557	pc	0.6000	ec	0.6429
wq	0.6118	netio	0.6720	CPU	0.6016	netio	0.6534

Table 4.5: Results from second step of forward selection, first alternative

Third step is evaluated with two fixed features, The two most significant features from earlier steps (smem and netio) and all the features in the data set in separate runs.

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
network	0.5777	CPU	0.6408	errlist	0.585	CPU	0.6389
pageing	0.5843	swap	0.6446	pageing	0.5891	pc	0.6431
swap	0.5877	pc	0.6507	fr	0.596	swap	0.6448
iowaait	0.5894	network	0.6538	sr	0.5967	network	0.6512
tps	0.5936	pageing	0.6538	network	0.5984	errlist	0.6525
sr	0.5936	sr	0.6542	iowaait	0.5993	pageing	0.6535
fr	0.5951	fr	0.6543	swap	0.6003	fr	0.6536
ec	0.5962	iowaait	0.6565	fcach	0.6042	sr	0.6537
sload	0.5966	wq	0.6579	tps	0.6048	sload	0.6552
fcach	0.5976	tps	0.6598	CPU	0.6057	wq	0.6565
CPU	0.5992	sload	0.6601	ec	0.6082	iowaait	0.6581
pc	0.6021	ec	0.6639	wq	0.6092	tps	0.6587
errlist	0.6031	fcach	0.6693	pc	0.6093	fcach	0.6592
wq	0.6037	errlist	0.6746	sload	0.6131	ec	0.6679

Table 4.6: Results from third step of forward selection

In this step the errlist feature scores highest in the standard data set and quite low in the vectorized data set. This is probably due to the naive nature of the data set which often has many zeros (errlist is zero if there are no hardware errors reported on the server).

Fourth step is evaluated with three fixed features, The three most significant features from earlier steps (smem, netio and errlist) and all the features in the data set in separate runs. Table A.1 can be found in the appendix. errlist is zero for some of the servers and thus do not add any value for the classifiers. The most significant feature is here fcach in the standard data set.

Fifth step is evaluated with four fixed features, The four most significant features from earlier steps (smem, netio, errlist and fcach) and all the features in the data set in separate runs.

Table A.2 can be found in the appendix. The most significant feature is here fcach, again in the standard data set.

Sixth step is evaluated with five fixed features, The five most significant features from earlier steps (smem, netio, errlist, fcach and ec) and all the features in the data set

in separate runs.
Table A.3 can be found in the appendix. This time tps in the standard data set scores barely higher than iowait.

Seventh step is evaluated with six fixed features. Here the evaluation value AUC for the features tps and iowait are very close, so we check both features in this step of forward selection. The values evaluated with tps can be found in table A.4 in the appendix and for iowait in table A.5. The observed result does not give any suprises, and we conclude that iowait is the most significant feature in the seventh step of forward selection.

Eighth step results in table A.6 in the appendix. As can be seen from the graphs in figure 4.3 and 4.4 the curves now levels out, indicating that we have found the set of most significant features for failure prediction. Adding more features for classifying will not give any substantial improvement of the prediction but it will just take a longer time to predict.

The graphs below shows evolution for the most significant features, measured as area under curve (ROC). Calculated as intermediate values for all sixteen servers. y-axis is AUC and x-axis is number of forward selection steps.

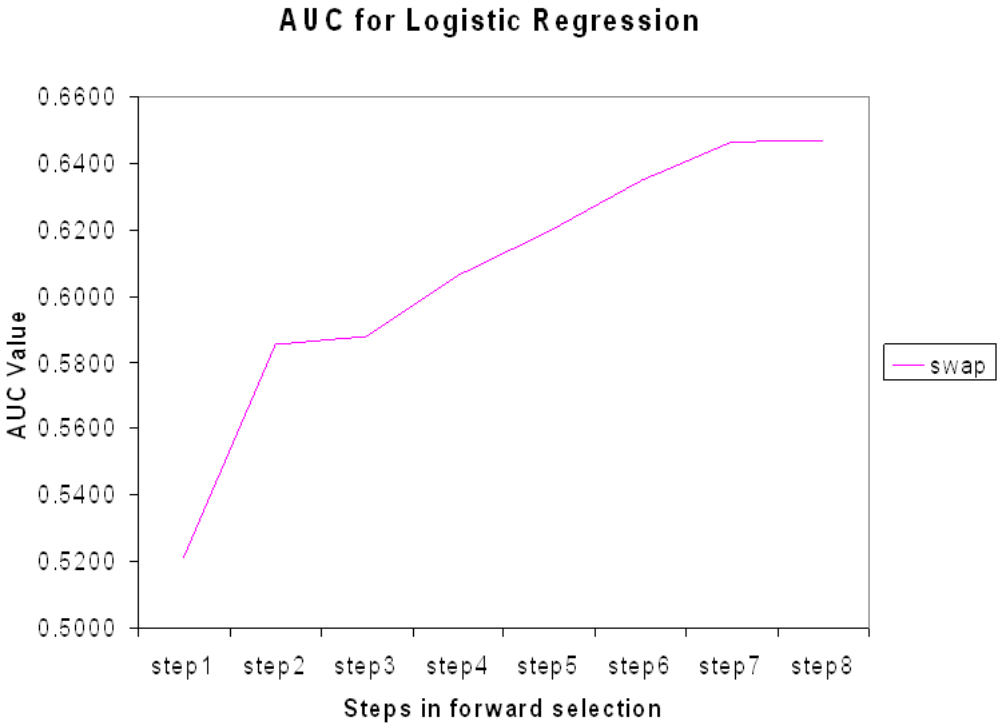


Figure 4.5: Forward feature selection with the Logistic Regression classifier

We observe that the curve in step two for Logistic Regression doesn't ascend at the same rate as for Naive Bayes. This is owing to the monotonous content in the feature errlist. The majority of the servers does not report any hardware errors, thus the value of the feature errlist often is zero. This makes it difficult for the Logistic Regression classifier to predict, and we often experienced program interruption probably due to one of the beta coefficients escaping towards infinity or because one of the values did not converge. We can conclude that Naive Bayes gives the best classification accuracy and needs two less steps in the search algorithm to find the optimal set of features.

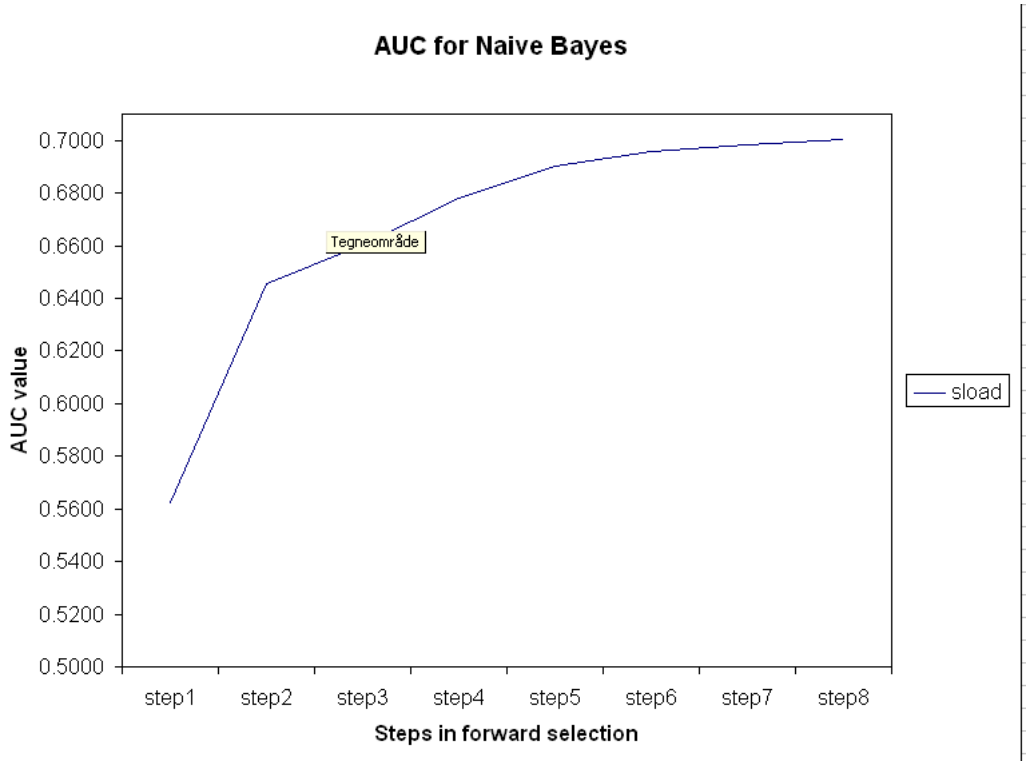


Figure 4.6: Forward feature selection with the Naive Bayes classifier

4.5.4 Differences between virtual servers and physical servers

Due to the fact that there are no failure-prone physical production servers in the PMS data we are not able detect any differences. The failure-prone production servers used in this thesis are all virtual servers. But we do know that the computer hardware used in the production environments are built to enable best possible up time. This is often done with a UPS, dual power supply, RAID solutions and monitoring equipment, all which is run in a temperature controlled dedicated server room. Optimized hardware in combination with the UNIX operating system which is known for stability, we can then assume that one of the reasons for using virtual servers is to segregate software solutions that are failure-prone due to software bugs. Instead of having a failure on a server with many services the services can run on separate virtual machines.

4.5.5 Classifier performance

The classification accuracy is in general very high, about 98 percent. Where classification accuracy is the percentage of correct predictions made by the model when compared with the actual classifications in the test data. However it is impossible to give a good measure of the classification accuracy for failure prediction because we do not know how many failures that are recorded in the performance monitoring data set, compared to manual shutdowns.
Examining the tables with the intermediate results from the experiments we observe that using the standard data set as input for the Naive Bayes classifier gives the best accuracy with the AUC equal to 0.70, which is less than one percent better than with the vectorized data set.
For the Logistic Regression classifier the accuracy is only about five percent lower, but it has another drawback, the fact that features with monotonous content caused the classifier to fail, probably due to one of the beta coefficients escaping towards infinity or since the values didn't converge.
The Forward stepwise feature selection improved accuracy from 0.67 to 0.70 measured with AUC for the standard data set and classifying with Naive Bayes. As can be seen from the table 4.2 and A.6, the search algorithm improved the accuracy with approximately four percent.

In order to evaluate the performance speed we added a timer function in a python script to calculate the time it takes to classify the input data. The times are calculated when classifying a year of performance data using Naive Bayes. The time it takes to classify a:

- standard data set is approximately 36 seconds and
- vectorized data set is approximately 25 seconds

We get nearly 30 percent reduction in time used to classify, when the data set is vectorized. Running the same test after applying feature selection we get the time it takes to classify a:

- standard data set is now approximately 12 seconds and
- vectorized data set is now approximately 5 seconds

We observe that the classifier uses one fifth of the time when applying feature selection and vectorizing the input data.

If we compare the advantage of classifying speed to the accuracy we observe that vectorizing of the input data set gives under one percent less accuracy but it increases the classifying speed with an order of five (with the feature search algorithm). With this information we find that, thus applying vectorized input and the feature search algorithm gives a great advantage when it comes to the computing load, and has little significance for accuracy.

Logistic Regression			Naive Bayes		
Host names	CA	AUC	Host names	CA	AUC
h00u001	0.9806	0.8539	h00u001	0.9806	0.8374
l00u001	0.9796	0.8541	l00u001	0.9796	0.8470
r00u000	0.9770	0.8602	r00u000	0.9769	0.8675
t00u001	0.9811	0.6094	t00u001	0.9811	0.5966
u00u001	0.9817	0.6200	u00u001	0.9816	0.6200
xspu139	0.9812	0.5611	xspu139	0.9812	0.5702
xspu141fo	0.9801	0.6487	xspu141fo	0.9803	0.7801
xspu151	0.9821	0.6925	xspu151	0.9817	0.7109
xspu157fo	0.9811	0.5935	xspu157fo	0.9809	0.5685
xspu166mv	0.9806	0.6467	xspu166mv	0.9807	0.6783
xspu167mv	0.9798	0.6102	xspu167mv	0.9802	0.6809
xspu168mv	0.9813	0.5703	xspu168mv	0.9813	0.5626
xspu169mv	0.9809	0.7673	xspu169mv	0.9789	0.6833
xspu183ar	0.9814	0.6472	xspu183ar	0.9814	0.5365
z00u000	0.9815	0.5854	z00u000	0.9815	0.5846
z00u038	0.9811	0.5883	z00u038	0.9811	0.6133
Intermediate value	0.9807	0.6693	Intermediate value	0.9806	0.6711

Table 4.7: Initial intermediate evaluation measures for the classifiers Naive Bayes and Logistic Regression calculated for all servers with a standard data set

According to the classification accuracy that is above 98% our novel prediction solution is well fitted to the problem but in our case, we have the outside effect of features that is influencing the output class we predict. Thus, we get to high classification accuracy.

4.5.6 Choosing a classifier

Initially we tested several classifiers. KNN gives acceptable classification accuracy but is far too slow. The two classifiers that stand out are Naive Bayes and Logistic Regression and to run in a production environment we will recommend the Naive Bayes classifier, because it is a little faster, and more accurate than the Logistic Regression classifier and because the Logistic Regression classifier has flaws that made it malfunction.

Chapter 5

Discussion

In the following section we discuss the obtained results for, feature selection, performance, precision and verify them in the last section of this chapter.

Analyzing our novel approach to predicting failures for UNIX system has shown that it is possible to find failure-prone symptoms, but how far are we from implementing the solution in a production environment? Through our review of related research we discovered that the most recent work in the area of failure prediction has proved that selecting the optimal set of features can give better predicting results than the choice of predicting method. Forward stepwise selection of the features proved that this is true to some extent, but in our case it gives more of a contribution to the classification speed. When preparing the performance data for the classifiers we observed that the UNIX shell script with the AWK utility for preprocessing the input data is impressively fast and flexible. The preparation of the data also revealed that the quality of the data is important to achieve good predictions.

5.1 Feature selection

In order to train the classifier for all the servers we made a generic set of features that can be used to classify performance data from any UNIX server. We used Forward stepwise feature selection to help remove the most irrelevant and redundant features from the PMS data, and to improve the learning models by:

- Alleviating the effect of the curse of dimensionality. In our case this is not a great

challenge because of the relatively low number of features. If more metrics are added to the PMS system it gives more of an advantage.

- Enhancing generalization capability, which is important with respect to making a generic set of features for all UNIX servers.
- Speeding up the learning process in order to achieve online prediction.
- Improving model interoperability.
- Helping us to acquire a better understanding of the performance data by telling which are the more important features and how they are related to each other. For instance the most significant feature is used server memory, which indicate that the server may have memory leak.

Even though we made a generic set of features, we discovered that servers used in the same application area (uarea) have quite similar evaluation measures when the performance data is classified. This points out that some application areas are more failure-prone than others and as a verification of our solution it indicates that the prediction works.

5.2 Performance

Preparing the data set we found that the AWK command used in the preprocessing has very good performance and powerful programming flexibility. The AWK utility performs with an impressive speed and prepares one year of performance data for a server in a few seconds.

In the first trials of the different classifiers we found that KNN was very slow for this task and was therefore ruled out. Testing the KNN classifier with one year of performance data for one server took approximately two hours, while the Naive Bayes classifier can do the same job in seconds. Evaluation of the classifiers showed that the Logistic Regression classifier performed surprisingly fast and use approximately the same time as Naive Bayes. To run in a production environment we will recommend the Naive Bayes classifier, because it is a little faster, and more accurate than the Logistic Regression classifier and because the Logistic Regression classifier has flaws that made it malfunction.

Applying feature selection and vectorizing the input data, we improved the performance for the Naive Bayes classifier by an order of five.

5.3 Accuracy

Examining the tables with the intermediate results from the experiments we observe that using the standard data set as input for the Naive Bayes classifier gives the best accuracy with the AUC equal to 0.70, which is approximately one percent better than with the vectorized data set. For the Logistic Regression classifier the accuracy is only about five percent lower. Features with monotonous content caused the Logistic Regression classifier to fail, probably due to one of the beta coefficients escaping towards infinity or because the value didn't converge.

The most important thing to improve the accuracy of the failure prediction is to improve the quality of the input data. This is because we know that some of the recorded failures are in fact not failures. Some recorded "failures" are just reboots after maintenance. To avoid recording manual shutdowns as failure, the PMS system needs enhanced functionality that can distinguish this in the database. There is also the possibility for a sudden power cut which can not be predicted. In such cases it should be sufficient with a manual function in the PMS to correct the error, because this happens very seldom.

The Forward stepwise feature selection improved accuracy from 0.67 to 0.70 measured with AUC for the standard data set and classifying with Naive Bayes. An important aspect with the Forward stepwise feature selection that arises is the fact that we do not know if the set of optimal features change if we change the time window for failure prediction. It is not very likely that this will happen but we do not know for sure. Also the disadvantages of forward stepwise selection can play a role in prediction accuracy because each addition of a new variable may render one or more of the already included variables non-significant.

5.4 Requirements

The requirements are met without any big challenges, however we only tested the Orange canvas and Python modules in Windows operating systems, but we expect this software to run without flaws in Linux.

5.5 Verifying the results

Using Naive Bayesian classifier and Logistic Regression classifiers we have shown that failure-prone patterns can be found. Logistic Regression to some extent shows the same results as Naive Bayes, but has inferior values. The two algorithms are based on completely different techniques:

- Naive Bayes with strong (naive) independence assumptions that assumes the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature
- and Logistic Regression which is a generalized linear model used for binomial regression that predicts the probability of occurrence of an event by fitting data to a logistic curve.

In addition we have tested the algorithms with ten fold cross-validation that splits the data into the ten folds by holding out the examples from one fold at a time; the model is induced from the other folds and the examples from the held out fold are classified.
Thus this indicate that the results are consistent and valid.

Chapter 6

Conclusion and further work

In this thesis we have investigated failure prediction methods for use in online systems, and in particular we have studied the domain of symptom monitoring and supervised pattern recognition with the Naive Bayes and Logistic Regression classifiers. Our focus has been to find a solution that can enhance an existing performance monitoring system with the capability of predicting upcoming failures.

In this thesis we propose a novel approach to online failure prediction for software failures, the solution has been formally presented, analyzed, and empirically tested.

We prepare the performance data set as binary classes for input to the classifiers. The preparation is accomplished with help of the AWK utility in a UNIX script. The solution proved to be surprisingly fast and well suited for this task.

The input data is optimized with the search algorithm Forward stepwise selection to help remove the most irrelevant and redundant features. Thus improving the learning model and finding a generic set of features for UNIX servers. Furthermore the model is evaluated with the input data in two formats, standard and vectorized. The vectorized input data gives approximately, one percent decreased accuracy, but on the other hand it reduces the time it takes to classify by an order of magnitude.

Several classifiers have been tested on the data set and found useless. The Naive Bayes and Logistic Regression classifiers are selected for the model. Both classifiers are evaluated and compared with each other. The Naïve Bayes classifier has approximately five

percent better accuracy and needs two less steps in the search algorithm to find the optimal set of features.

6.1 Conclusion

We have observed that our classifiers are able to classify the failure symptoms with high overall accuracy. However this is based on the fact that there is some uncertainty due to the quality of the input data, because some of the failure symptoms are in fact not failures but actually reboots after maintenance.

Furthermore our experiments demonstrate that open source software can be used to make an enhanced Performance Monitoring system with online failure prediction. We believe that our research gives new insight to the field of online failure prediction and is a good foundation for further research.

6.2 Further work

The method presented in this thesis has demonstrated its ability to classify symptom monitored data with very high accuracy, however there is a need for further improvements.

An important task in order to achieve better failure prediction is to improve the quality of the input data. In order to rule out records that aren't failures, but a reboot after for instance maintenance, the Performance monitoring system must be notified that this is not a failure. Another possibility is to implement a manual solution for correcting the stored performance data.

We observed that the classifiers evaluation measures for servers running the same type of applications are quite similar. This indicates that the feature selection optimalization will improve if the procedure is automated for groups of servers running the same type of applications.

Bibliography

[1] Laprie, j. c. and avizienis a., dependable computing. from concepts to design diversity. Proceedings of the IEEE, volume 74(5):629 to 638, May 1986.

[2] Hoffmann G. A. Failure prediction in complex computer systems: A probabilistic approach. Shaker Verlag., 2006.

[3] A. Agresti. *Building and Applying Logistic Regression Models*. John Wiley and Sons, Inc., Hoboken, NJ, USA. doi: 10.1002/9780470114759.ch5, Department of Statistics, University of Florida, Gainesville, Florida, USA, 2 edition, 2006.

[4] A. Andrzejak and L. Silva. Deterministic models of software aging and optimal rejuvenation schedules. In 10th IEEE/IFIP International Symposium on Integrated Network Management (IM 07). 159-168., 2007.

[5] R. P. H. Hunter S. Trivedi K. Vaidyanathan K. Castelli V., Harper and Zeggert W. Proactive management of software aging. In IBM Journal of Research and Development 45, 2 (Mar.), 311-332., 2001.

[6] Tsai P. Chung Y. Cheng F., Wu S. and Yang H. Application cluster service scheme for near-zero-downtime services. In IEEE Proceedings of the International Conference on Robotics and Automation. 4062-4067., 2005.

[7] Naive Bayesian classifier www.wikipedia.org.

[8] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. Technical report, 2004.

[9] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, pages 861–874, 2006.

[10] Maren Lenk Felix Salfner and Miroslaw Malek. A survey of online failure prediction methods, March 2010.

[11] Steffen Tschirpke Felix Salfner. Error log processing for accurate failure prediction. WASL'08 Proceedings of the First USENIX conference on Analysis of system logs, USENIX Association.

[12] Greg Hamerly and Charles Elkan. Bayesian approaches to failure prediction for disk drives.

[13] Tibshirani R. Hastie T. and Friedman J. The elements of statistical learning: Data mining, inference, and prediction. Springer Series in Statistics. Springer Verlag., 2001.

[14] TRIVEDI K. S. HOFFMANN G. A. and MALEK M. A best practice guide to resource forecasting for computing systems. IEEE Trans. Reliab. 56, 4 , 615-628., December 2007.

[15] Asial Corporation http://www.asial.co.jp/ [Online].

[16] Python Software Foundation http://www.python.org/psf/.

[17] Virtualization Concepts International Business Machines.

[18] Faculty of Computer Laboratory of Artificial Intelligence and Slovenia. Information Science, University of Ljubljana.

[19] Rao T. Scott S. Leangsuksun C., Liu T. and Libby R. A failure predictive and policy-based high availability strategy for linux high performance computing cluster. In The 5th LCI International Conference on Linux Clusters: The HPC Revolution. 18-20, 2004.

[20] Vaidyanathan K. Li L. and Trivedi K. S. An approach for estimation of software agingin a web server. In Proceedings of the Intl. Symposium on Empirical Software Engineering, ISESE 2002. Nara, Japan., 2002.

[21] Sivasubramaniam A. Jette M. Liang Y., Zhang Y. and Sahoo R. Bluegene/l failure analysis and prediction models. In IEEE Proceedings of the International Conference on Dependable Systems and Networks (DSN 2006). 425-434., 2006.

[22] Hilbe Joseph M. *Logistic Regression Models*. Chapman and Hall/CRC Press, ISBN-10: 1420075756 / ISBN-13: 978-1420075755, 2 edition, May 2009.

[23] K nearest neighbor www.scholarpedia.org.

[24] G. Choi R. Lal. Error and failure analysis of a unix server., November 1998.

[25] P. E. Hart Richard O. Duda and D. G. Stork. *Pattern Classification*. John Wiley and Sons, 2 edition, 2001.

[26] Felix Salfner and Miroslaw Malek. Hidden semi-markov model based failure prediction. In IEEE Proceedings on 26th International Symposium on Reliable Distributed Systems (SRDS 2007)., 2010.

[27] Open source database ORACLE CORPORATION [Online].

[28] D. Turnbull and N. Alldrin. Failure prediction in hardware systems. Tech. report, University of California, San Diego., 2003.

Appendix A

Appendix

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
network	0.5967	CPU	0.6587	pageing	0.5922	CPU	0.6481
pageing	0.6022	swap	0.6628	swap	0.5937	pc	0.6524
swap	0.6062	pageing	0.6652	network	0.5963	swap	0.6534
iowaait	0.6088	pc	0.6684	fcach	0.6019	network	0.6597
sr	0.612	network	0.6722	sr	0.6035	pageing	0.6616
tps	0.6125	sr	0.6724	CPU	0.6036	sr	0.6623
fcach	0.6129	fr	0.6726	fr	0.6043	fr	0.6624
fr	0.6133	iowait	0.6751	iowaait	0.6052	sload	0.6643
ec	0.6151	wq	0.6768	ec	0.6079	wq	0.6651
sload	0.6158	tps	0.6782	wq	0.6098	iowaait	0.6662
CPU	0.618	sload	0.6782	tps	0.6112	tps	0.6676
pc	0.6207	ec	0.6816	sload	0.6119	fcach	0.668
wq	0.6227	fcach	0.6849	pc	0.6128	ec	0.6757

Table A.1: Results from fourth step of forward selection

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
pageing	0.6160	CPU	0.6685	pageing	0.6051	CPU	0.6577
swap	0.6195	swap	0.6726	swap	0.6075	swap	0.6607
network	0.6200	pageing	0.6760	network	0.6128	pc	0.6632
sr	0.6216	pc	0.6794	sr	0.6128	network	0.6665
fr	0.6234	network	0.6818	fr	0.614	pageing	0.6681
iowait	0.6246	sr	0.6821	iowaait	0.617	sr	0.6686
tps	0.6315	fr	0.6821	CPU	0.6217	fr	0.6687
sload	0.6371	iowait	0.6850	tps	0.6272	wq	0.6709
CPU	0.6372	wq	0.6855	sload	0.6275	iowaait	0.6731
ec	0.6410	tps	0.6865	ec	0.6308	sload	0.6733
wq	0.6418	sload	0.6904	wq	0.6314	tps	0.6735
pc	0.6443	ec	0.6920	pc	0.6331	ec	0.6845

Table A.2: Results from fifth step of forward selection

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
pageing	0.6232	CPU	0.6701	swap	0.6329	CPU	0.6566
swap	0.6394	pageing	0.6785	sload	0.6334	pc	0.6644
iowait	0.6404	pc	0.6816	network	0.6334	swap	0.674
pc	0.6408	swap	0.6833	pageing	0.6341	network	0.683
sload	0.6413	sr	0.6953	pc	0.6354	sload	0.6837
network	0.6414	fr	0.6953	tps	0.6375	pageing	0.6846
tps	0.6454	network	0.6956	CPU	0.6375	fr	0.6849
sr	0.6467	sload	0.696	fr	0.6382	sr	0.685
fr	0.6479	wq	0.6964	sr	0.6385	wq	0.6867
CPU	0.6504	iowait	0.6971	iowaait	0.6448	tps	0.6884
wq	0.6545	tps	0.6982	wq	0.6464	iowaait	0.6887

Table A.3: Results from sixth step of forward selection

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
pageing	0.6239	CPU	0.6734	swap	0.6365	CPU	0.6607
swap	0.6437	pageing	0.6811	pageing	0.6387	pc	0.6691
iowait	0.6457	pc	0.6848	network	0.6388	swap	0.6777
pc	0.6458	swap	0.6864	sload	0.6398	network	0.687
network	0.6474	wq	0.6981	pc	0.6401	sload	0.6877
sload	0.648	fr	0.6982	sr	0.6434	fr	0.6884
sr	0.6507	sr	0.6982	CPU	0.6443	pageing	0.6885
fr	0.6512	network	0.6987	fr	0.645	sr	0.6885
CPU	0.653	sload	0.699	iowaait	0.6458	wq	0.6892
wq	0.655	iowait	0.7003	wq	0.6485	iowaait	0.6907

Table A.4: Results from seventh step of forward selection. Case 1

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
pageing	0.6273	CPU	0.6724	swap	0.6451	CPU	0.6606
swap	0.6464	pageing	0.6809	network	0.6455	pc	0.6684
tps	0.6468	pc	0.684	tps	0.6463	swap	0.6779
pc	0.6482	swap	0.6856	pageing	0.6467	sload	0.6844
network	0.6501	wq	0.6973	pc	0.6468	network	0.6872
sload	0.6503	fr	0.6975	sload	0.647	wq	0.688
sr	0.6518	sr	0.6975	sr	0.6505	pageing	0.6882
fr	0.6527	network	0.6979	wq	0.6518	sr	0.6884
wq	0.6532	sload	0.6982	CPU	0.652	fr	0.6885
CPU	0.6548	tps	0.6996	fr	0.6521	tps	0.6901

Table A.5: Results from seventh step of forward selection. Case 2

Standard				Vectorized			
LR		NB		LR		NB	
Features	AUC	Features	AUC	Features	AUC	Features	AUC
pageing	0.6273	CPU	0.6749	swap	0.6445	CPU	0.6633
swap	0.6469	pageing	0.6828	pageing	0.6462	pc	0.671
pc	0.6492	pc	0.6862	network	0.6465	swap	0.6804
network	0.6506	swap	0.6878	pc	0.6468	network	0.6894
sload	0.651	wq	0.6986	sload	0.6482	sload	0.6898
sr	0.6522	sr	0.6996	sr	0.6504	wq	0.6899
fr	0.6533	fr	0.6996	wq	0.6514	fr	0.6907
wq	0.6537	network	0.7001	CPU	0.6526	pageing	0.6908
CPU	0.6557	sload	0.7002	fr	0.6526	sr	0.6908

Table A.6: Results from eighth step of forward selection