CHALMERS

GÖTEBORGS UNIVERSITET

Machine Learning for Classifying Cellular Traffic

Learning how to Predict Overloads in the Cellular Network

Master's thesis in Software Engineering and Technology

ISABELLE FRÖLICH

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017:06

Machine Learning for Classifying Cellular Traffic

Learning how to Predict Overloads in the Cellular Network

ISABELLE FRÖLICH


CHALMERS

GÖTEBORGS UNIVERSITET

Department of Computer Science and Engineering
Software Engineering and Technology
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden

Machine Learning for Classifying Cellular Traffic
Learning how to Predict Overloads in the Cellular Network
ISABELLE FRÖLICH

Supervisor: Miroslaw Staron, Department of Computer Science and Engineering
Examiner: Regina Hebig, Department of Computer Science and Engineering

Master's Thesis 2017:NN
Department of Computer Science and Engineering
Software Engineering and Technology
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Machine Learning for Classifying Cellular Traffic
Learning how to Predict Overloads in the Cellular Network
ISABELLE FRÖLICH
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Today's cellular network is ever growing, making the need for a mechanism that can identify overloads greater each day. In this report a design science research is conducted showcasing the possibilities to use the classification machine learning algorithm naive Bayes to identify signaling overloads in a cellular network node. The research shows that naive Bayes can be used to successfully identify the greater majority of the possible overloads that could occur in a cellular node.

Keywords: Machine learning, naive Bayes, cellular network

Acknowledgements

I would like to thank Ericsson for providing me with the data that I needed to carry out this research and a special thanks to Patrik Willard for coaching me the whole time through the research. I would also like to thank Bengt Strömberg for his feedback and support. And finally I would like to thank my supervisor, associate professor Miroslaw Staron for guiding me though the thesis.

ISABELLE FRÖLICH, Gothenburg, Sweden, 2017

Glossary

AP Application Processor - the payload of the administrative activities..

DP Device Processor - the payload of the user activities..

PM Performance Measurement - measures the performance of the SGSN-MME node and contains a combination of AP and DP data..

SGSN-MME Serving GPRS Support Node and Mobility-Management Entity - the node in the cellular network of interest in this report..

Contents

List of Figures

List of Tables

1

Introduction

Today there are millions of mobile devices connected to the internet each day and the amount keeps increasing [1]. Managing cellular traffic in a satisfying way to keep the network's users content is an important task for many telecom companies [1]. The cellular network consists of a number of different nodes that route the right data packets to the right phone and allows the phone to stay mobile without losing any packets. In any given situation these nodes can become overloaded if it receives extreme amounts of signals. This can happen at any time and place for a number of different reasons, e.g. if many people try to connect to the network at the same time. Today no mechanisms are in place that can identify the overloads before they happen.

The signals being sent to and from a node can be saved in logs that contain performance measurement (PM) data. PM data contains thousands of instances of thousands of variables making it impossible for humans to interpret. The problematic consequences of an overloaded node is currently minimized by different predefined actions that decrease the functionality of the node. One of the predefined measures is to measure the CPU load of the node and if the CPU load is too high certain calls can be dropped from the network. Only when the node is already overloaded is it visible to the people responsible.

The overload of a node is often caused by an extreme amount of signaling and if the spikes in signaling and other critical patterns could be identified the overload might be mitigated. Even if the problem is discovered only as much as an hour beforehand, the appropriate action could be taken. For example a technician could be sent to the location to mend the problem. No amount of testing can hinder a network from becoming overloaded in an extreme situation. It is therefore of interest to try a new approach, which could possibly circumvent the issues with the overloads.

In recent years it has become more popular to use machine learning to solve problems that involve a large amount of data [3]. More machine learning libraries and guidelines have become available online which can facilitate the implementation of the algorithms [3]. If one of the available machine learning techniques could be used to classify data and predict possible overloads in a cellular network it might be possible to prevent the overloads from happening at all.

As machine learning is good for handling large amounts of data it would be suitable for finding patterns in the PM data that humans can not find. If the pattern recognition is successful it could be used to send out a warning when the node is about to be overloaded. By learning how to recognize a potentially problematic patterns the computer could warn the administrator when the node needs backup and technicians could be called in. This could decrease the downtime of the node to a bare minimum which would be beneficial for the telecom companies and create more reliable products for their end-users. If such an approach is successful it could be a step on the way to create a node that has no unplanned downtime.

Many books on machine learning have been published [2] - [7] and papers have applied machine learning in many different fields [10] [11]. But, there does not exist much research about error handling in combination with machine learning and especially not error handling in network components. As machine learning is a novel problem solving technique the effectiveness of the chosen algorithms in this type of problem is evaluated during the research and can in the future be compared to other techniques.

The report aims to describe how to solve the problem by developing a prototype that uses machine learning and draw conclusions from the findings. The work will be done in collaboration with the telecom company Ericsson to gain hold of real world PM data. The report is structured as follows; In chapter 2 the component of interest, the SGSN-MME and the PM data is examined more closely. An overview of the machine learning field and the different classification algorithms is also introduced in that chapter. In chapter 3 the methodology and research questions answered in this report can be found. After that, in chapter 4, the actual implementation of the algorithm is showcased. In the following chapter the results of that implementation is presented. The two last chapters discusses the results and draws conclusions from the discussion and the results.

2

Background

The background section will first describe the machine learning field as a whole in order to later describe the different classification algorithms that have been considered in this paper. It will also describe the component in the cellular network that the research concerns, the SGSN-MME node, and the data that it produces.

2.1 Machine Learning

In literature machine learning and pattern recognition are well defined concepts [2]-[7]. There are a number of different algorithms that are used in machine learning. Commonly these algorithms are categorized into two approaches; unsupervised- and supervised learning [5]. Unsupervised learning means that the task is performed without any previous reference points. In supervised learning, on the other hand, the task is performed with some already known facts or examples. There is also a middle ground where the algorithm uses a combination of the two approaches called semi-supervised learning [5]. The nature of these algorithms is explained more closely later in this chapter to provide a rationale for which type of algorithm will be used in this thesis. A general picture of how machine learning works in comparison to how normal programming works can be seen in figure 2.1.

Supervised learning utilizes an external resource called a *teaching set* as a reference to optimize the algorithm [12]. This teaching set entails both the initial inputs and the corresponding ideal outputs. With the teaching set the algorithm, learns how to map input values to output values in the best way [7]. Supervised learning algorithms work by first letting the machine know what input maps to which output and then, with the help of that knowledge, the algorithm cam estimate the output. Supervised learning algorithms are validated by entering input into the algorithm for processing and then controlling that the output matches what is desired. Then, the given output can be measured to estimate how effective the chosen algorithm is. In this research, a supervised learning algorithm will be used to predict which patterns will cause an overload in a node.

As mentioned, unsupervised learning algorithms do not receive a complete teaching

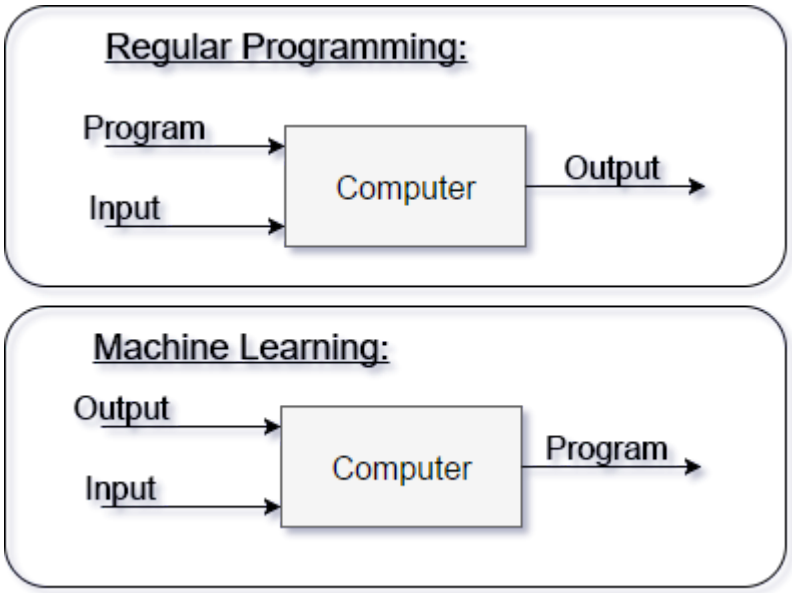

Figure 2.1: The general concept of how machine learning works versus how regular programming works [12].

set to use in the learning phase. The teaching set used by unsupervised learners only contains input without any corresponding labels that specifies the expected output[5]. This means that no defined answers exist that the machine can map the input against. As nothing is known about the input, no predictions about the data can be made, instead the data is described or grouped in different ways[7]. To measure the effectiveness of the chosen unsupervised learning technique the given output and the expected output has to be evaluated [7]. There are a number of different algorithms of this kind but they all have in common that they only describe the data. One of the most common unsupervised learning algorithms is *clustering* which groups attributes with similar patterns together [12]. Another unsupervised algorithm is *association rules* which is used for detection of patterns [12]. It is not possible to provide these types of algorithms with the wanted output which makes them useful for tasks were the label or category of the data is not known [7]. Unsupervised learning is rarely used when there exist examples where the answers are known and there exists no unsupervised classification algorithms [7]. Unsupervised learning is instead often used for data mining [12].

The combination of the two earlier mentioned approaches is called semi-supervised learning which is an approach that has evolved in the computer science community in recent years. Semi-supervised learning uses a teaching set, as the previously mentioned approaches, but the set only contains a very limited amount of examples of input and its corresponding valid output. This requires a bit more of the algorithm and of the data, as more assumptions has to be made to draw conclusions about it. But, as the example data can be time consuming to put together, in certain cases, for example when it comes to audio and video data, it is of interest to minimize the amount of manual work. With the semi-supervised approach the algorithm has a reference but not as much work needs to be done to provide the reference. In

some cases it is even more advantageous to have less data as too many data points can make the data too complex to process. [5] Semi-supervised learning will not be evaluated further in this thesis as there exists a lot of teaching data.

As mentioned these approaches are widely used in literature and they are supertypes of a number of different techniques that can be used in machine learning [7]. Supervised learning is a supertype for, among others, the concepts of classification and regression. While under the supertype unsupervised learning there are the concepts of clustering and dimensionality reduction [7]. Semi-supervised learning can include some of the concepts of both supervised- and unsupervised learning [5]. All of these machine learning techniques are used to perform certain tasks and they are in turn supertypes of the algorithms that are used to help solve these tasks. As classification algorithms only exist under the supervised learning supertype; only supervised algorithms will be considered further.

One of the mentioned supervised learning techniques is *regression* [7]. Regression algorithms learn how to estimate an output value for a given input value by fitting the values to form a prediction function, i.e. numeric prediction [3]. The other mentioned supervised learning technique, classification, is used to identify which class a given data point belongs to [3]. The machine learns this technique by using the algorithms to analyze the mapping between the data in the teaching set and the corresponding valid output. This is often done by looking at the attributes of the input data to recognize patterns that might occur [12]. These algorithms include, for example, *decision trees* [6], *neural networks* [3], *naive Bayes* [12] or *nearest neighbor* [12] that are used in different situations. As this research focuses on classification, only classification algorithms will be evaluated further.

2.2 Classification Algorithms

In this section a number of machine learning algorithms used when performing classification tasks are described. As there are more classification algorithms than can be described in this report only the most common ones will be mentioned in section 2.2.1 and 2.2.2. The algorithms are often useful in similar situations and the effectiveness of an algorithm when solving a certain problem can only be measured after it is implemented[9]. Therefore, no conclusions about their effectiveness can be showcased in this sections. Instead, it focuses on describing the algorithm's characteristics and common areas of use. The algorithms are classified into two groups; white box methods and black box methods [12]. White box methods provide models that are more or less easily understood. Black box methods, on the other hand, provide models that are nearly impossible to understand.

Problems with classification algorithms includes overfitting and underfitting data. Overfitting a model means that the model follows the variations in the data very well. Each peak and valley is modelled by the algorithm and not one data point is missed by the model. But the problem of overfitting is that humans often make

mistakes, for example, mistakenly entering the wrong value. This mistake is then reflected in the model which will affect the reliability of the algorithm negatively. Besides mistakes other factors can influence the data negatively, for instance, faults in the measuring systems. This undesirable data is called noise and can slip through if a model is overfitted. The opposite problem is an underfitted model which does not model the data well and can miss small, but important, patterns. Both problems are equally bad, and it is therefore important to find the right balance that creates a representative model of the data. Examples of both underfitting and overfitting can be seen in figure 2.2 where the nearest neighbor algorithm is affected by both problems. K stands for the number of examples considered when placing a new object.[12]

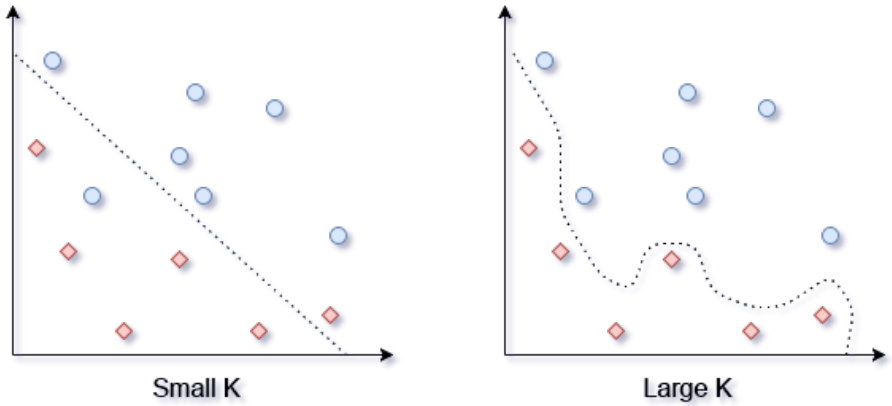

Figure 2.2: The problem of underfitting and underfitting the nearest neighbor algorithm [12].

As mentioned it is important to evaluate the performance of the algorithm to know how well it worked in a certain situation, one way of doing this is to measure the accuracy of the chosen classification algorithm. A measurement of the accuracy of a classification algorithm can be computed in a number of different ways but only measures for classification tasks with two classes are considered. The most simple way to measure the accuracy is to compare the number of correctly predicted classes to the number of instances in a test set [12]. This type of accuracy is calculated with this formula:

$$(TN + TP)/(TN + TP + FN + FP)$$

Where TN and TP stands for true positives and negatives and FN and FP stands for false positives and negatives [12][16]. A positive class is the class is the class that is of most interest and the negative class is the class of less interest. For example if we are predicting if an email message is spam or not the positive class is the spam class and the negative class is the non-spam class.

A true negative happens when an instance is classified correctly and belongs to the class of low interest or the negative class [16]. A true positive is also an instance classified correctly but which belongs to the class of interest or the positive class [16]. A false positive happens when the algorithm wrongly classifies an instance in the class that that indicates that something of interest has happened. A false

positive is often accepted as opposed to a false negative as a false negative results in an important event being missed. If the number of correctly predicted classes is close to the total amount of instances, the accuracy is good [12]. A similar way of estimating the accuracy is to include the number of false positives in the equation, then the formula is:

$$(TN + TP + FP)/(TN + TP + FN + FP)$$

This measurement is used in situations where it is of high importance to avoid a false negative but a false positive does not entail any far-reaching consequences [12]. This measure can sometimes be of greater interest than the first mentioned measurement of accuracy as it better illustrates how many events of interest that are missed.

Even though the above mentioned accuracy models provide a measure of the performance of the algorithm other measures could add even more insight about the performance of the algorithm [12]. The problem with the accuracy model is that if the existence of instances of one class is very low the algorithm will not have to predict that class very often and the accuracy can therefore seem to be better than it is. If, for example, the accuracy is 99% and one of the classes only appears in 1% of the instances the accuracy does not say anything about the performance of the algorithm [12]. Therefore other complementary measures have been developed to evaluate the performance of the algorithms. One of these measures is the Matthews Correlation Coefficient which measures if the prediction could have happened by chance or if the prediction is deliberate [20].

Matthews correlation coefficient is one of the most common correlation coefficients used when evaluating the performance of binary machine learning classifiers [18][17][19]. Matthews correlation coefficient can take on any value between -1 and 1, where -1 indicates that the predictions have a negative correlation with the accurate results, 0 that the predictions have been assigned randomly and have no correlation with the correct classes and 1 that the predicted classes fully correlate with the accurate classes [20]. The equation for the coefficient is:

$$(TP * TN - FP * FN)/\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}$$

where TP, TF, FN and FP are abbreviations for the same concepts as explained earlier [21].

2.2.1 White Box Methods

White box methods are a type of machine learning algorithms that create a model of the underlying data that can be understood by a human. This model can be useful when interpreting the results of the given machine learning algorithm as it provides information about why the results look like they do. The model can also be used to find new and interesting patterns in the data that was not visible beforehand. White box methods can be other algorithms than classification algorithms, but this

section only focuses on white box algorithms that are classification algorithms as well. [12]

Decision Trees

Decision trees are well defined classification algorithms [12] which are sometimes called classification trees [7][14]. A decision tree is, just as the name suggests, a tree structure where each node in the tree represents a decision that has to be made in order to traverse down the tree[12]. Decision trees are binary, i.e. each parent node has a maximum of two children and they can handle both numerical and other attributes[14]. They operate by taking the decision with the highest abstraction level first, i.e. the factor that seems to be the most influencing one [3]. This initial decision will divide the data into two groups. Then the algorithm continues down the tree making more and more fine grained decisions, partitioning the tree each time[5]. Finally, it arrives at the smallest concluding decisions, called the terminal or leaf node. If the decision is final is determined by a given criteria, for example; the tree size or the percentage of nodes with the same class [12]. An example of this process can be seen in figure 2.3 were the rings marked with a "n" are the nodes and the squares marked with a "l" are the leaves. The conditions where the tree splits are the division criteria.

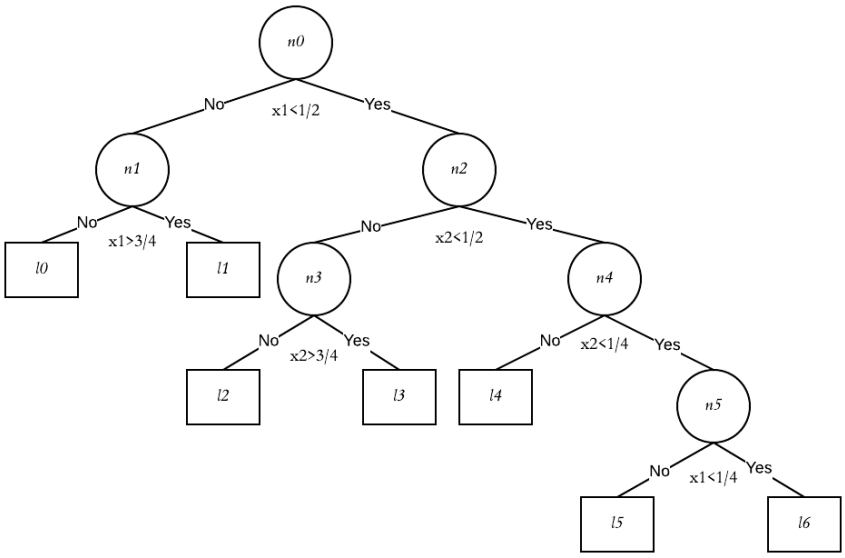

Figure 2.3: An example of how a decision tree works [14].

The decision tree is quite easy to follow with a clear outcome that makes it suitable for applications in processes that require a transparent workflow [4]. There is a number of different algorithms that use decision trees, the most prominent one being the C5.0 algorithm [12]. This algorithm contains features to minimize the risk of overfitting or underfitting the decision tree as both of these problems are common when using decision trees [12]. A technique specific to this algorithm that minimizes these two problems is called pruning and entails removing branches that are not supposed to affect the classification [14]. Decision trees are commonly used with

problems where the solution needs to be transparent. This lead to the algorithm being one of the most popular in the banking industry where transparency is a requirement [12]. For example, the algorithm can be used when performing online purchases that requires a persons bank credit to be pre-approved [12].

Classification Rules

The classification algorithm *classification rules*, sometimes called *rule learners*, is a variation of the decision tree that instead uses different rules as splitting criteria which are formulated in a standard programmatic way with "if" and "then" statements. The rules are used as decision makers to conclude when the tree should split and are comprised of two components: antecedents and consequents. Antecedents are pre-conditions that have to be met, i.e. the "if" statement. The antecedents describe values that the attributes can take on and are followed by a consequent. The consequent is the outcome if the antecedents are fulfilled, i.e. the "then" statement. Classification rules are even easier to understand than decision trees as they clearly describe why a split was made. As opposed to decision trees classification rules are separate from the model and do not have to be applied in a certain order. Classification rules are suitable for the same tasks as decision trees are, they can for example be used when identifying reasons for hardware failures. [12]

Nearest Neighbor

Nearest neighbor algorithms group objects from the teaching set with the same classifier together and looks at how their attributes are similar, an example of this can be seen in figure 2.4 [3]. When a new object is introduced the algorithm looks at the known examples and their attributes to determine in which classifier the new object should be placed [3]. Nearest neighbor algorithms commonly use Euclidean Distance to measure how “close” the object’s attributes are to the attributes in the teaching set [5]. The class the input belongs to can be determined by looking at only the example which is closest to the object. Another option is to look at several of the closest examples and choosing the class that is represented most often [5]. How many examples that should be considered in the nearest neighbor algorithm is a trade-off [12]. Too few examples can cause noise or errors in the data to affect the outcome and the result will be overfitted. Choosing too many examples can cause the data to be underfitted which causes the algorithm to ignore trends that are not as obvious as big trends. The amount of examples considered using the nearest neighbor algorithm usually varies between 3-10 examples [12].

This type of algorithm is called a *lazy learner* as it does not create a model of the data that it uses in the classification process [12]. Instead, it simply consults the teaching set to classify the input. This makes the algorithm very quick in the learning phase. However, the downside of this is that it makes the algorithm relatively slow in the classification phase [14]. The nearest neighbor algorithm does not, as opposed to many other algorithms, make any assumptions about the distribution of the data, which means that it can be used with most types of distributions and types of data [12]. As mentioned the nearest neighbor algorithm looks at the characteristics of the attributes and groups the input with the group that it is most similar to [12].

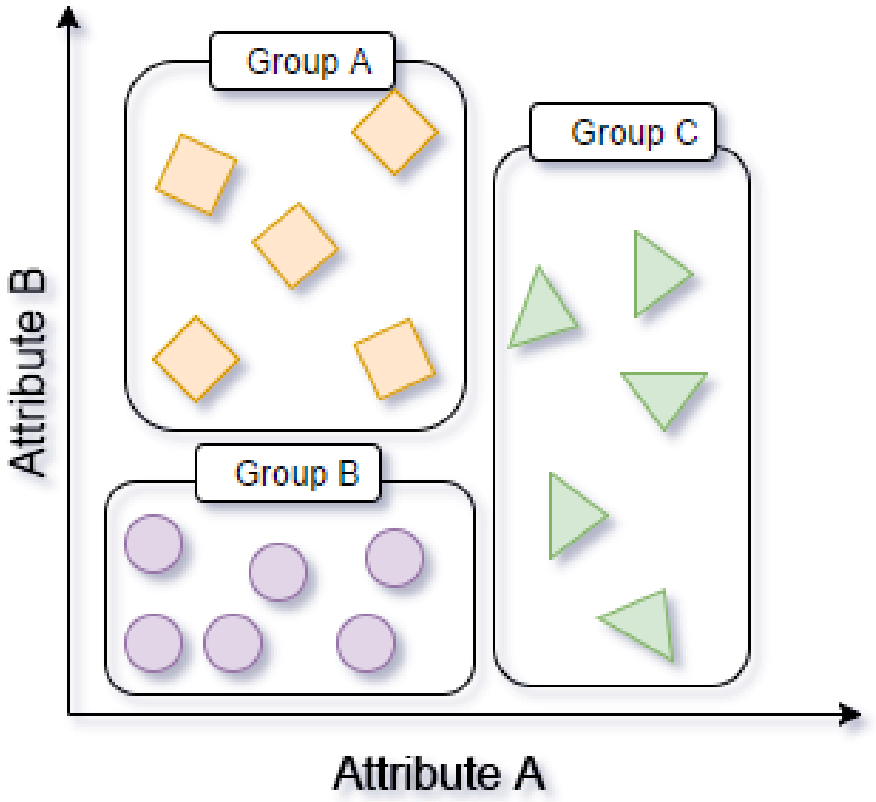

Figure 2.4: An example of how the nearest neighbor algorithm works [12].

It is therefore useful for face recognition and similar tasks as it is able to group a multitude of complex attributes together and create clear groups of them [12]. It does on the other hand not perform well with attributes were the boarders of the groups are fuzzy and hard to define [12].

Naive Bayes

Naive Bayes is a form of probabilistic learning that uses probability to calculate the chance of a certain outcome, called an event [12]. As it is one of the supervised learning algorithms it uses a teaching set to learn how the data works [14]. The teaching set is used to create a model that is representative for the data. After the teaching phase it is ready to receive input [12]. To map the input to the right group it groups known events in the model with patterns similar to the input, which are identified by examining their respective attributes [12]. The group will then be used to calculate how many of the instances in that, that led to a certain outcome. The percentage calculated is the chance of the outcome happening again given the provided input. This is calculated with the help of the Bayes theorem seen in figure 2.5.

Naive Bayes relies on a number of assumptions for the algorithm to work, this is why it is called “naive”. For example it assumes that the different attributes are independent of each other, i.e. that they do not affect one another, it also assumes

that all attributes are equally important [14]. This means that no attribute should affect the outcome more than another attribute [12]. Even though the algorithm assumes this it can still, in some cases, provide good results if the assumptions are broken [12]. This has made naive Bayes a very popular algorithm that suits a multitude of different classification problems. If the assumptions are however met the algorithm is even more suitable for solving the problem at hand and is very quick in doing so. It also has the benefit of providing accurate results in situations were some data is noisy or incomplete [12]. The ease of use has made the naive Bayes algorithm popular when classifying junk mail. It can also be used when creating weather reports and for detecting anomalies.

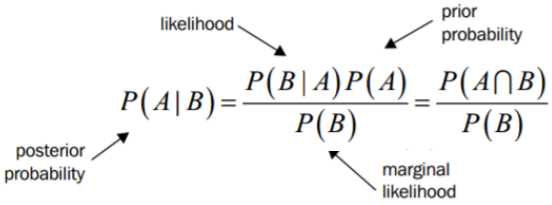

Figure 2.5: Bayes theorem - The underlying formula for naive Bayes.

Bayes theorem uses three different probabilities to calculate probability of an event happening given certain attribute values as can be seen in figure 2.5. The chance of an event A happening given the attribute value B is called the posterior probability. The posterior probability is calculated using the likelihood, the prior probability and the marginal likelihood. The likelihood is the probability that attribute value B occurred with event A in the past. The prior probability is the chance that event A will happen. And the marginal likelihood is the chance that the attribute will take on value B. [12]

To calculate the probabilities the algorithm uses the standard deviation (calculates how much the values vary from the mean) and the mean of each attribute in a class. If there are two classes the mean and standard deviation needs to be calculated for each attribute in both cases, i.e. there will be 2*(number of attributes) calculations. These values are used to calculate the probability of a certain value belonging to a class. If the posterior probability exceeds 50% the input should be assigned to the class that specifies that event A will happen. If the chance is under 50% it should be assigned to the class that specifies that the event will not happen.[12]

Naive Bayes can classify continuous numeric variables in two ways, the first one being by binning the data. Binning means that the data is sorted into intervals that can be of varying size, but this method leads to information loss [12]. The other method is to use the Gaussian probability density function which assumes that the underlying distribution is normal [15]. This way of classifying instances is sometimes called the Gaussian classifier and uses the probability density function to calculate the probability of a variable belonging to a certain class [15]. The function uses the value of one variable x, the mean of the variables in the investigated classμ and the standard deviation of the variables in the investigated class σ to calculate the

probability of that value belonging to the class with the following function:

$$(1/\sqrt{2 * \pi * \sigma^2}) * e^{-(x-\mu)^2/2\sigma^2}$$

2.2.2 Black Box Methods

There are a many other algorithms that can perform classification tasks besides the ones mentioned in the previous section. The algorithms described above are relatively easy to understand and the models they create are transparent and accessible. Other methods are not as transparent and comprehensible which gives them the name black box methods. Two of these algorithms are *Neural Networks* and *Support Vector Machines* (SVM) . Both these algorithms produce models that are complex and hard to understand, which is why they were classified as black box methods. However, even though they are hard to understand, they can be very accurate when modeling certain real world problems. Neural Networks come from the neuroscience field and uses concepts first discovered when studying how the brain works to perform complex machine learning tasks. Support Vector Machine uses a multidimensional space to map features to the correct output. [12]

Support Vector Machines

Support Vector Machines can be used for both classification tasks and to create linear regression models for numeric predictions as well as for many other problems but, are most popular when classifying patterns. They have for example been used when classifying text, like recognizing what language a text is written in. SVM uses what is called a hyperplane to separate data into groups with the same classifier. This hyperplane can stretch over multiple dimensions and is therefore useful when modeling real world problems that has a large amount of features. The hyperplane is chosen by finding the plane that separates the data and has the maximum margin on both sides of the data. This hyperplane is called maximum margin hyperplane (MMH) and is important as it is the best way of separating the data so that future data points are classified correctly. The support vectors are the points in the dataset from both classes that are closest to the MMH. The support vectors allow the algorithm to be memory efficient even with large amounts of data, as only the vectors need to be saved for future reference. [12]

The mentioned MMH is only applicable when the data can be separated by a straight plane. If this is not the case another method that allows data points to be on different sides of the plane is used. In this case the so called "cost" of the data point is calculated by estimating the distance from the proposed line. The plane is placed so that the cost of the data points on the different sides is minimized. As SVMs are used in such a wide variety of problems they use a number of different ways to estimate how the training data should be separated. They can also use kernels to trick the algorithm to view the data in a linear way. The SVM uses a number of different techniques to model the data and have proven to be very accurate. The algorithm is not very sensitive to noise nor overfitting and has recently gained a lot of support in the form of easy to use libraries for many different languages. It can

also model problems with a very large amount of features, even though the training process istime consuming. The downside is that the model that is created by the algorithm is very hard to interpret, which makes it difficult to draw conclusions from it. [12]

Neural Networks

Neural Networks can also model a number of different problems just as the SVMs and they can even be used for unsupervised learning. The network started out as a model of the human brain that was used to showcase how a brain functioned. Later research evolved and the algorithm was used to model more versatile problems. The algorithm has for example been used for speech recognition in phones and turning handwriting into speech. The neural network can model very complex problems. This is due to the fact that a network can contain several hundreds of artificial neurons, which each take a number of weighted variables as input. These neurons, with the help of a function, are used to calculate the output.

The actual network is defined by three different properties; an activation function, a network topology and a training algorithm. The first characteristic of a Neural Network is its activation function. The activation function transforms the input and then forwards the output to the rest of the network if a certain threshold value is met. There are a number of different threshold functions that can be used to determine if and how the output should be forwarded. One of these functions is the unit step activation function, which is a binary function that forwards the value if the threshold is met, and does nothing if it is not met. The most commonly used activation function is the *sigmoid function* which allows non-binary output in a range from 0 to 1. [12]

The network topology is also an important part of the Neural Network. The topology is for example determined by how many neurons and layers that are present in the network. For an easy classification task only one layer might be needed, with one neuron for each input feature. But for more complex tasks multiple layers, called hidden layers, can be added to increase the computational power. The topology is also defined by the direction of the data flow, which is either forward or both forward and backward. The last property that defines a Neural Network is the training algorithm. The most developed and used training algorithm is backpropagation which iterates over the neurons to estimate the weights of the input. The technique is slow and produces a model that is too complex to understand. However it has proven to be a very accurate and versatile machine learning algorithm. [12]

2.3 SGSN-MME Node

The SGSN-MME is one of the nodes that forms the cellular network and is the node of interest in this research. The SGSN-MME is a node in the cellular network that, among other things, handles the mobility of a phone connected to the network. The SGSN part of the acronym stands for Serving GPRS Support Node (GPRS in

the acronym stands for General Packet Radio Service). SGSN handles the mobility of the entities, for example mobile phones, connected to the 2G/3G network. The MME part of the acronym stands for Mobility-Management Entity. The MME handles the mobility of the entities attached to the 4G network. This node handles incoming traffic and routes the traffic onward in the network.

Figure 2.6: Picture of one version of the SGSN-MME node [1].

The SGSN-MME is made up of a number of identical cards placed in a box as can be seen in figure 2.6. The cards are numbered with odd numbers ranging from 1 up to the maximum of the number of cards possible in that specific node. These cards handle the traffic that comes though the SGSN-MME and a balancing algorithm controls that each card handles the same amount of traffic to make sure that a single card does not get overloaded while the other ones are empty. Each card contains multiple cores which are split up into groups that handle different types of traffic. The amount and type of data handled by the node at a certain point in time can be saved as a node dump that contains information about the node and its traffic. These node dumps, or logs, are only created if something goes wrong in the node or if someone has explicitly instructed the node to create a node dump. The node dump contains the PM data which is produced by the node. More about the PM data is found in the next section.

2.4 PM-Data

PM data consists of numerous numerical counters and variables that keep track of the signal traffic in the SGSN-MME node. A counter can only be incremented and counts the number of times a certain type of signal arrives in the node. For example; if an idle phone tries to connect to the node a counter that keeps track of the number of idle attachment attempts is incremented. The variables, also called gauges, on the other hand can both be incremented or decremented. The gauges show, for example, the number of attached units at a certain point in time. The

PM data produced by the SGSN-MME node is written in a node dump, or a log, that can be saved and used to analyze the signaling patterns of the node. Below, in figure 2.7, a draft of what such a node dump could look like is presented.

	A	B	C	D	E
1	Measurement Time	bssgpAttResumeProc	bssgpAttSuspendProc	bssgpBssInitiatePtpBvcReset	bssgpBssInitiateSignalingBvcReset
2					
3	2015.11.21 00:00:02	1487873021	3227100370	178931	73
4	2015.11.21 01:00:02	1488288633	3228095291	178937	73
5	2015.11.21 02:00:02	1488505008	3228639886	178962	73
6	2015.11.21 03:00:02	1488653710	3228973259	178969	73
7	2015.11.21 04:00:01	1488780719	3229220839	179023	73
8	2015.11.21 05:00:01	1488906653	3229453678	179043	73
9	2015.11.21 06:00:01	1489053752	3229738046	179043	73
10	2015.11.21 07:00:02	1489294324	3230292599	179105	73
11	2015.11.21 08:00:02	1489754100	3231469760	179232	73
12	2015.11.21 09:00:02	1490413378	3233152395	179344	73
13	2015.11.21 10:00:02	1491351866	3235405885	179420	73
14	2015.11.21 11:00:02	1492575233	3238190934	179692	73
15	2015.11.21 12:00:02	1493889332	3241226001	179838	73
16	2015.11.21 13:00:01	1495269119	3244473325	180018	73
17	2015.11.21 14:00:02	1496582216	3247639998	180118	73

Figure 2.7: Sample of a node dump from 2015 where the node was overloaded after three days.

There are two types of signals that the gauges and counters keep track of; user plane payload (DP - Device Processor) and control plane payload (AP - Application Processor). DP is the signals that the user is accountable for, for example when the user uses the cellular network to consume data by streaming a video. AP is the signals that are produced in the background and is more of an administrative nature. One example of this administrative signals is keeping the phone connected to the network even when it is moving around but not in use which allows the phone to be mobile without loosing connection. This type of signalling is not only affected by how much a user uses their phone, but also by more physical aspects such as how much space there is between mobile towers in a given area. PM data shows both AP and DP signaling.

The PM data consists of 4010 counters and gauges which all keep track of different signals. As can be seen in figure 2.7 the counters and gauges, both referred to as *variables*, have names that explain what they are measuring. For example, the third column in figure 2.7 counts the number of attached entities with processes that are suspended. Another possible variable name is for example: AttIdle1_1_E(AP). The initial acronyms explain what the variable is measuring, for example the amount of attached idle entities. The numbers following the explanation shows the number of the card that the information applies to. The number of the card is odd numbered from 1_1 to 1_23 in the node dumps which they are also named in the physical entity. The single capitalized letter after the numbers show which type of traffic is measured. E stands for 4G traffic, U stands for 3G and G stands for 2G. The brackets contain which type of data is measured by this variable, which is AP, DP or both.

The values of the variables are added to the PM-data ones every hour but the PM data is only saved if an overload occurs. If and overload occurs, the point in time when the overload happened can be gathered from the logs which include alarms with time stamps. The alarms can be used to identify when the overload happened and hence also which variable values that are measured before and after the overload. The variable values that are recorded during the hour the overload is happening is hereafter called the *transition instance*. The instance of variable values that is recorded the hour before the overload has happened is called the *pre-transition instance*.

A database at Ericsson stores some of the node dumps which contain the PM data produced by SGSN-MME nodes used by real customers. The logs produced by the node shows the overloads but, as it contains a multitude of different variables and is several megabytes in size, it is difficult to analyze it and find the overloads. It is also problematic to locate the elements of interest. It is therefore not practical for a human to interpret the signaling patters that can be gathered from the logs. The data could however be analyzed by a machine if it knew what it was looking for. If the logs that are produced when a node is overloaded are used as a teaching set for a machine learning algorithm, it could be used to find patterns that human eyes can not find.

2.5 Related Work

There are many different studies that make use of the mentioned algorithms and apply them to their respective fields. Machine learning has for example been used in social sciences [10], finance [11] and many other fields. Many studies on the application of machine learning has also been made in the telecommunications and internet fields were many different classification algorithms has been applied to a variety of problems [24][22][23]. A common machine learning classification task is to classify different types of internet traffic [24] [22]. This problem has, for example, been solved with decision trees, naive Bayes and Neural Networks. Another problem in the same field is the classification of malicious code in the network [23]. This problem has also been solved using naive Bayes and decision trees.

There are other similar work that uses machine learning to predict the amount of traffic, for example, in cloud computing, in distributed systems and in different applications [25][26][27]. This work also focuses on predicting peaks in the CPU load, but in other fields and with other data. A number of these problems are solved with Gaussian processes and Bayesian learning, often in a combination of the two, with successful results [25][26]. It has also been solved using Euclidean distance [27]. Although there exists a lot of research about machine learning and much research has been conducted to investigate how to predicting overload, not much research has been done on predicting overloads in the cellular network.

Several reports has explored the difference between the algorithms discussed in sec-

tion 2.2. The Third IEEE International Conference on Data Mining discusses the difference of the classification algorithms naive Bayes, SVM and decision trees [28]. The report states that there are no significant differences in the prediction accuracy between the three, even though the accuracy varies slightly between them [28]. Another report discusses the difference between naive Bayes, SVM and nearest neighbor when applied to another classification task[29]. This report declares that naive Bayes and SVM provided better predictions than nearest neighbor but that naive Bayes needed more training data than SVM and SVM needed time consuming manual tuning to perform well [29]. Several other comparing studies has also been performed with varying results, but no study concluding the best algorithm for classifying overloads is found.

3

Methodology

To evaluate if it is possible to use machine learning when classifying patterns in the PM data, a design science research will be used. The first section in this chapter introduces the research questions and how they will be answered. The second section will give a specification of how the design research will be performed and what steps is taken during the research.

3.1 Research Questions

To start the research it is important to choose an algorithm that can classify the data instances as good, no overload has occurred, or bad, an overload has occurred. This will be done by evaluating the different algorithms considered in the literature review. After the evaluation an algorithm that can use a teaching set to classify data patterns will be chosen. The classes will be one of two: data with a high probability of overloading the node or data that has a low probability of overloading. This will be answered with the following research question:

Which machine learning algorithm is suitable to use when predicting if a node in the cellular network is about to be overloaded?

When this question is answered an examination of the PM data has to be done to make sure that machine learning can be used to classify patterns in this data which could include preparations of the data in different ways. The second question is therefore:

How can machine learning be used to identify the patterns in the PM data that causes CPU overloads in the SGSN-MME node?

When an algorithm is been chosen it will be used to create a prototype that shows whether it is possible to classify the data that produces the high CPU loads. This algorithm is evaluated to draw conclusions on whether it is suitable for the problem or not, which will answer the question:

Is it suitable to use a classification algorithm to predict overloads in the cellular network?

3.2 Research Methodology

The research is organized into three steps to follow the design science research that is specified below. The first step is only performed once but the two last steps are iterated over two times. The first iteration separates itself from the second as only one node dump is used, while the second implementation uses four node dumps. The difference is that with the one node dump the files does not have to be combined and formatted in a different way. The three steps are specified below:

- The research starts out with an evaluation of all the different classification machine learning techniques. This is done by the literature review in chapter 2 which is carried out with the structure of the PM data in mind. The different algorithms suitability for classifying the data is evaluated. The outcome of this step is that the machine learning algorithm Naive Bayes is chosen.

- The second step is that an analysis of the possibilities of using that machine learning technique when classifying patterns in the PM data that precedes critical problems is carried out. This is done by manually evaluating the PM-data to get a better understanding of what it looks like. After the manual inspection is performed a script that formats the data in a suitable way for it to work with the naive Bayes algorithm is developed.

- After the algorithm is chosen and the data is transformed into a suitable format that works with the Naive Bayes algorithm a proof-of-concept is developed. The proof-of-concept is used to showcase how a machine learning technique can be used to classify PM data and includes measurements of accuracy and correlation to the underlying data so that the algorithm can be evaluated.

Design science research focuses on showcasing how a novel technique can be used in practice to gain more understanding of a certain phenomena [13]. This research method is chosen as it uses a practical implementation to build the knowledge base. By designing a prototype that uses machine learning it is possible to analyze how well the algorithm fit the problem and if it should be used in future similar problems. The most important part about design science research is to produce new knowledge or to strengthen the knowledge about a particular problem. The design science research process can be seen in figure 3.1.

The first step in the design science research is the *Awareness of Problem*. This step includes creating awareness of a problem which for example can be a gap in the knowledge base of the research field. This can be done by identifying a new problem or a applying a novel solution to a known problem. The outcome of this step is the proposal and the problem statement which is developed in the beginning of the

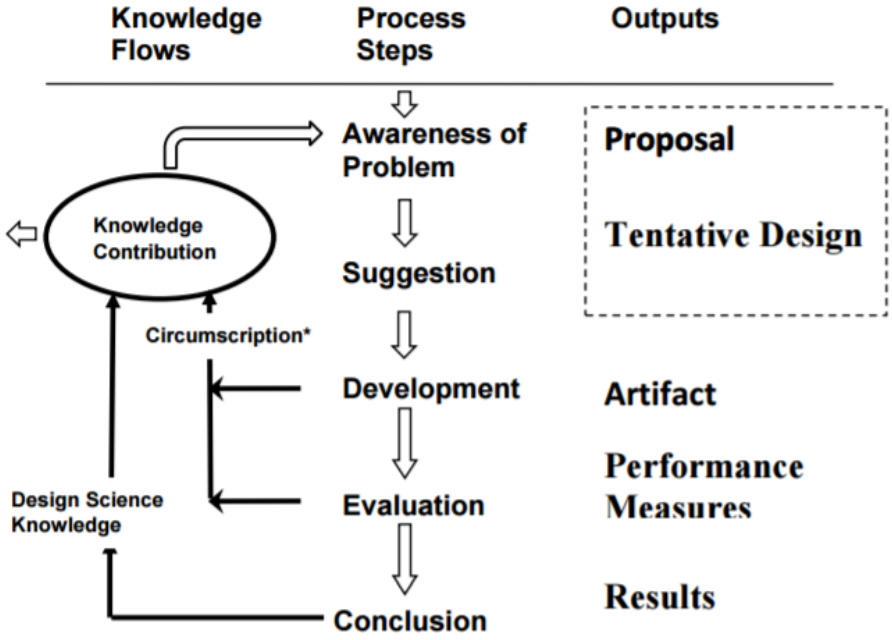

Figure 3.1: Process when performing design science research [13].

research. The essence of this output is the research questions, which can be found in the previous section. The problem statement in the proposal describes the problem and is used in the next step, suggestion. In the suggestion phase a suggestion of how the problem should be solved is created. The proposal and the tentative design which is the outcome of the suggestion phase are closely connected. This is because the design is started already in the proposal and is then more closely specified in the suggestion phase. This phase in the research is performed with the help of a literature review. The literature review is done in order to suggest a suitable algorithm for this problem and is specified below in a subsection.

When the literature review is done and the suggestion of the appropriate algorithm is made the development phase can start. In this case the development phase is the application of the algorithm on the given problem. The development phase started with the data being transformed into a suitable format. After that a solution with the Naive Bayes algorithm is developed. When the implementation is done the algorithms suitability and the outcome of the implementation is evaluated, which is the evaluation step. The evaluation includes metrics that measure how well the implementation works and what is gained from the implementation. The evaluation phase is described in the results and discussion chapter in this report. From the evaluation and the measurements conclusions can be drawn and possible future improvements can be suggested. This is the conclusion chapter in this report. The outcome of the process should be new knowledge that contributes to the research field.

3.2.1 Literature Review

In order to get broad knowledge of the machine learning field as a whole and to be able to decide on a suitable algorithm for the problem, a literature review is conducted. In this research the literature review is not the main part of the research, hence the literature review is relatively small and focused on specific algorithms. The databases Google Scholar and Chalmers Library are searched for phrases such as "Machine Learning" and "Pattern recognition". Books and articles that describes the machine learning field are used to get a broad overview of the different algorithms and the field as a whole. After that the most mentioned algorithms are investigated more closely. As this report focuses on classifying data only classification algorithms are covered in more detail.

One of the classification algorithms mentioned in section 2.2.1 and section 2.2.2 sections will be used for the research in this thesis. Different algorithms are suitable for different problems and even if one technique is chosen it is not obvious if that algorithm is the most efficient one [9].

3.2.2 Preparation of Data

The data used with the first implementation of naive Bayes is real world data received from Ericsson. This node dump contains 4020 different counters and gauges that affected the CPU load during the time the dump was created. The values of these counters and gauges, hereafter referred to as variables, were recorded the year 2015 from 21 November 00:00:02 to 30 November 23:00:02. The variable values were recorded each hour which counts to a total of 240 instances of each variable. The values of many of these variables did not change during the time that the node produced the data. To apply the machine learning algorithm naive Bayes the data has to vary in time as described in section 2.2.1. This is because the algorithm cannot calculate the standard deviation of a variable that does not vary and as the standard deviation is essential to calculating the probability static variables has to be removed.

The previously described node dump, hereafter called the *full node dump*, is the only found node dump that contains all available variables. No other node dumps that contain all variables besides the full node dump is found in the database, but three other node dumps that could be used are found. The three node dumps are from the year 2015 and are seen as suitable as they contain a good mix of recorded data from both before the overload and after the overload. This makes the nodes suitable as the algorithm needs to learn both the patterns of an overload and the patterns when nothing goes wrong. These three node dumps contain both types of events and the point in time of the of the overload could also be identified with the help of the logs of alarms that are stored in the system.

The original node dumps are in the form of regular text documents, the .txt format,

with the vertical bar, "|", separating the columns in a row and a newline separating the rows. The data is turned into an excel, .csv, spreadsheet before the process of reducing the variables takes place. The text document is converted into a .csv file by entering it as input in an excel sheet where the variables are separated from eachother by specifying the vertical bar as a separator. After the data is converted a column at the end of the file needs to be added that specifies which class each row of data belongs to. The classes are always added in supervised learning to teach the algorithm which type of data is bad and which is good [12]. A row of data in the csv file contains all existing variables in that file and their values at a certain point in time and is called an *instance*. Each instance in a file needs to belong to a class which is used to label the data and is one of two classes: no overload has occurred (good = class 0) or an overload has occurred (bad = class 1). The events; bad or good, are mutually exclusive and an instance can hence only belong to one of the available classes.

The exact point in time when the node starts overloading can be gathered from the recorded alarms that come with the node dump, which in the first node dump is after approximately one third of the time had passed. The full node dump only contains one *pre-transition instance* and one *transition instance*, i.e. the first instance that is labeled as 1 as described in 3.2.2, as each node dump only covers one overload occasion. When the time the overload occurs has been discovered the instances are assigned classes, which means that 35% of the 240 instances in the full node dump are labeled with the class 0 and 65% of the instances are labeled with the class 1. The *second node dump* is chosen because it has a fair amount of variables, but it only contains bad events which means that it contains no transitions. This node dump is recorded the year 2015 from 21 November 00:00:02 to 30 November 23:00:01 just as the full node dump. It has 462 variables in common with the full node dump and also contains 240 instances. As the overload alarms happen before the the node dump starts all of the instances are labeled with class 1, the bad class, and hence the node dump does not contain a transition instance.

The *third node dump* has 306 variables in common with the full node dump and the second node dump and it contains 273 instances. The data of the third node dump is also recorded the year 2015 from 21 November 00:00:00 to 2 December 10:00:01 and contains one transition from class 0 to 1. In this node dump the problems occur at the end of the file which means that most of the instances in this file are labeled as good. Approximately 95% of the data is labeled as good, class 0, and 5% of the data is labeled as bad, class 1. Finally the *fourth node dump* is labeled. This node dump also has 306 variables in common with the other files and contains 1171 instances. This file also contains both good and bad events, approximately 10% good events and 90% bad events, which means that it contains one transition instance. The node dump is recorded from 21 November 00:00:02 to 8 December 10:00:01 the year 2015 and the problems start 24 November around 13:00:00. All mentioned node dumps record the variable values each hour, with few exceptions when a recording that hour is missed. The implementation of the script that formatted the data is found in chapter 4.

3.2.3 Development

The chosen algorithm is decided to be one of the classification algorithms described in 2.2 and therefore also a supervised learning algorithm. Naive Bayes is recommended in many books as a classification algorithm that has produced good results in the past [12][2][14]. Naive Bayes is also described as useful when evaluating many variables simultaneously and the variables are treated with the same importance. This is interesting as many small signaling patterns can be combined to create big patterns that can be of importance. Therefore naive Bayes is chosen as the machine learning algorithm to use with this problem.

The teaching set that the algorithm is fed with is real world node dumps that has been saved from previous node failures. These node dumps are explained in greater detail in the section 3.2.2 The first node dump is used to make sure that naive Bayes can be used in this type of problem and with the implementation described in chapter 4. The first implementation using the full node dump becomes the first prototype. The exact implementation of the first prototype of the implementation of the naive Bayes algorithm can be found in chapter 4. After the initial implementation is done the second implementation iteration takes place. The second implementation focuses on enabling the algorithm to analyze several node dumps. This is done by providing the algorithm with more data and validating that it works in the same way with the new data.

The algorithm is run 200 times for each combination of files. The first 200 runs will only include the full node dump, the next 200 runs will include the second node dump and the full node dump, the third 200 runs the third, second and full node dumps and the last runs will use the fourth, third, second and full node dumps. For each run the accuracy and the Matthews correlation coefficient will be calculated. The accuracy is measured in two ways, one measurement calculates the amount of correctly classified instances and one measurement calculates the same thing but includes the false positives. The values of the correlation coefficient and the accuracies are recorded each run and the mean of these measurements are used as the final measurements which shows the performance of the algorithm with the different combinations of files. The amount of correctly classified transition instances is also measured in two ways. One measurement identifies the amount of correctly identified overloads when they happen and one measurement shows the amount of identified overloads at least an hour before they happen. As mentioned these measurements use the transition instance and the pre-transition instance to calculate the amount of these instances that are correctly identified. The three files together contain three transition instances and three pre-transition instances.

3.2.4 Evaluation

When validating the prototype 75% of the data will be used to train the algorithm, which is approximately the same as three node dumps. The data which will be used

to train the algorithm is randomized to make sure that instances from each files are used and to not introduce any bias. The remaining 25% of the instances will be used to test how well the algorithm works. The measurements that will be used to evaluate the algorithm are the accuracy measurements and the the Matthews correlation coefficient. As stated in section 2.2 the measurement value -1 states that the algorithm's predictions have a negative correlation with the underlying data, which means that the predictions are opposite of what they should have been. 0 indicates that the classes have been assigned by random which is the worst value as it indicates that the predictions have nothing to do with the underlying data. Anything above 0 indicates that the predictions correlate with the data, where 1 means that they fully correlate. Anything above 0.5 will be seen as a good result and the closer the result is to 1 the better the algorithm is.

The accuracies are also used as a performance measurement to evaluate the algorithm. The amount of bad instances out of the total amount of instances is 65% which means that the accuracy of the algorithm at least needs to exceed 65% for the accuracy to say anything about the performance of the algorithm [21]. Therefore anything above 70% is seen as acceptable, 80% is seen as satisfactory and 90% is seen as a valuable classifier. The accuracy measurement that includes the number of false positives can be used to calculate the number of false negatives, i.e. how many overloads that are missed. Preferably the amount of missed overloads is below 10%.

The time saved can be estimated by investigating the amount of transitions from the good class, 0, to the bad class, 1, are correctly identified. As previously indicated the files contain variable values with time stamps an hour apart from each other. At the moment there exists no foresight when predicting the overloads, the baseline is therefore zero. If the prediction can save any time at all this is time gained. The downside of the prediction is that the time stamps are relatively far apart but alarms can occur between the time stamps. It is therefor hard to predict how much time is saved, if the time is not exactly an hour. It is only possible to estimate how many of the transitions are identified correctly and hence that time has been saved, but not how much time is saved.

4

Implementation

The implementation section describes in detail how the naive Bayes algorithm is implemented with the PM data in the node dumps. The first section describes the first iteration where naive Bayes is implemented with a single node dump file. The second section describes how the implementation of naive Bayes is done with several node dump files.

4.1 First Implementation

The first iteration of Naive Bayes uses the only node dump that contains all of the 4020 possible variables as input. This file is made up of 240 instances each containing variable values for all of the variables. An example of such an instance of PM data, including headers and time stamps, from the first node dump can be seen in figure 4.1 which illustrates the data before it is formatted. The first implementation step is to format the data to suit the naive Bayes algorithm which leads to a minimization of variables. This is done by writing a script in Python that takes a csv file as input and turns it into a list. From this list all constant variables are removed by comparing the values in the same variable column to each other. If the values do not differ from one another the entire column with that variable is removed as static variables can not be used with the naive Bayes algorithm [12].

Measurement Time	bssgpAttResumeProc	bssgpAttSuspendProc	bssgpBssInitiatePtpBvcReset	bssgpBssInitiateSignalingBvcReset
2015.11.21 00:00:02	1487873021	3227100370	178931	73

Figure 4.1: An example of four different variable values in a single instance of PM data before it has been formatted.

The script also formats the data in a way that is suitable for the algorithm by firstly removing the headers, which are not part of the data. Secondly the initial column, which only shows the current time and day, is removed as this column does not provide any value. As mentioned in chapter 3 a column of classification values is added in the end of the csv file and this column is kept intact through the whole process. The classification column is not changed or removed even if the values of the classification are static, e.g. all instances belong to class 0. After the data is

formatted and all of the constant variables are removed only 2110 variables are left in the list which is a reduction of 1910 variables from the original list. When the process is done the list is again turned into a csv file stripped of all unnecessities. A draft of what the stripped file looks like can be seen in figure 4.2.

2,51E+08	2,5E+08	1,46E+09	1556617382	1454557785	1447688988	0
2,51E+08	2,5E+08	1,46E+09	1556746765	1454678748	1447809565	0
2,51E+08	2,51E+08	1,46E+09	1557095379	1455005185	1448136383	0
2,52E+08	2,51E+08	1,46E+09	1557942143	1455795168	1448929097	0
2,52E+08	2,51E+08	1,46E+09	1558984526	1456763600	1449902791	0
2,52E+08	2,51E+08	1,46E+09	1560173020	1457865141	1451008693	0
2,52E+08	2,51E+08	1,46E+09	1561515246	1459111279	1452253116	1
2,52E+08	2,51E+08	1,46E+09	1562963664	1460457481	1453593057	1
2,53E+08	2,52E+08	1,47E+09	1564539162	1461905693	1455040261	1

Figure 4.2: A draft of the first node dump after formatting is performed.

After the formatting the remaining 2010 variables are split into a training set and a test set which are randomly assigned data instances based on a specified split ratio of 0.75. This leads to 75% of the formatted and labeled data instances in the first node dump being used to train the naive Bayes algorithm. An image of the split of this full node dump can be seen in figure 4.3 The algorithm is then trained by building a model of the underlying data of the training set, which essentially is a set of summaries of standard deviations and mean values for all of the instances belonging to each class. An example of what a summary can look like can be seen in figure 4.5 where the standard deviations and means of five variables has been calculated for both classes.

```
The file contains 2010 variables
The split ratio is 0.75
Split 240 rows into train=180 and test=60 rows
```

Figure 4.3: Illustrates how the full node dump is split into a training set and a test set.

The set of summaries is then used to predict the classes of each of the instances in the test set, the remaining 25% of the data. The probability of a certain variable value belonging to a certain class is calculated with the help of the Gaussian probability density function with the help of the set of summaries developed in the training phase. The standard deviation and mean from the current investigated class and variable is used. The probability of that variable value belonging to that class is then multiplied with the probability of the next variable's value belonging to the same class. This is done until all variable values probabilities have been multiplied and

the product of this multiplication is the probability of the entire instance belonging to that class. The probability of the same instance belonging to the other class is then calculated in the same way. An example of the probabilities of six different instances belonging to the two different classes can look like, can be seen in figure 4.4.

```
{0.0: 8.82911339074125e-90, 1.0: 3.5569791742584945e+56}
{0.0: 3.194019569073674e-89, 1.0: 9.950913864683535e+54}
{0.0: 4.6728148593378215e-107, 1.0: 5.3013811999595006e+48}
{0.0: 1.21310341252711108e-103, 1.0: 4.889748650896757e+52}
{0.0: 1.8824737535741933e-101, 1.0: 6.679832886599169e+41}
{0.0: 2.2976439222775303e-89, 1.0: 3.2579093556635547e+55}
```

Figure 4.4: An draft of what the probability assignment can look like.

Because large amount of variables present the probability of an instance belonging to a class decreases with each variable. For example if the probability of a variable value belonging to a certain class is 0.9, a very high probability, and the same probability of all other variable values belonging to that same class is also 0.9 the total probability will be $1.0652 * 10^{-}92$. Therefor all of the the probabilities are amplified with the same value for the effects to be visible in the final calculated probability of the entire instance belonging to the class. The probabilities are organized in a similar way as the summaries with one probability value for each class.

```
{0.0: [(1488080827.0, 293882.0635425034), (1613597830.5, 2281837174.2843304), (178934.0,
 4.242640687119285), (4479173351.5, 3528865.181879367), (118279.0, 163985.13362497222)],
 1.0: [(1488648286.75, 112732.61037036563), (2421654730.0, 1614414816.9823077), (178980.
75, 28.359301824974466), (4486637365.0, 1346205.0353137148), (18663.5, 19862.69980809926
7)]}
```

Figure 4.5: An example of what a summary created in the learning phase can look like.

To predict which class the instance belongs to the probability of the instance belonging to one class is compared to the probability of the instance belonging to the other class. The class with the highest probability is then predicted to be the correct class of the instance. The predicted classes of the test set are then compared to the correct classes and the accuracy of the model is computed. The accuracy is computed in two different ways, one percentage measurement which measures how many of the instances that are classified correctly and one percentage measurement which measures the amount of the correctly classified instances together with the false positives. The second mentioned measurement is useful as it can be used to calculate how many false negatives that occur, i.e. the amount of overloads that are missed with the algorithm. Finally the Matthews correlation coefficient is calculated with the formula described in section 2.2 which describes the chance of the classification happening randomly.

The number of correctly identified transitions, i.e. when the class label switches from no overload (class 0) to overload occurring (class 1), is also measured two

times. To create these measurements the location of the transition is first identified. If the only transition present in the full node dump is labeled with class 1 the overload is classified correctly. The second measurement calculates if the instance before the overload was classified as an overload. As the instances that are used as a test set are set randomly the algorithm is run 200 times and the accuracies, Matthews correlation coefficient and transition percentages are calculated each time for a correct value for the evaluation. The mean of these values are used to evaluate the accuracy and the coefficient to give a picture of how well the algorithm worked with the problem.

4.2 Second Implementation

The implementation of the naive Bayes algorithm with several files is done in a similar way, with the main difference being that the preparation of the data beforehand is done differently. The second formatting script is also a Python script but which is implemented to take a list of strings pointing to the location of the node dumps as input. The implementation takes the first csv file the string is pointing to and adds the variables of this file to a list. The csv file the second string in the list is pointing to is then opened and the variables in that file are added to another list. The variables in the new list are iterated and compared to the variables in the original list. If the new variable's header matches a header added in the original list the values of the variable in new list are appended to the matching variable's list in the original. This is done for all variables in all csv files in the list of strings.

After the files are combined all of the headers are removed as they no longer serve a purpose and the inital time stamp column is also removed. The second step of the process is then to remove the variables from the original list that do not have as many values as the variable with the most values. This step is important as the naive Bayes algorithm can not evaluate instances that do not have values for all of the variables in the list. The variables that are of the maximum length are therefore used as a baseline and all other variables that do not meet the requirement are removed. The last step of the implementation is the same as in the implementation of the script that works with a single file and that is to check whether the list contains static variables and if so remove them. The list is thereafter turned into a csv file again that works with the same implementation of naive Bayes as mentioned in the previous section. The only difference in the csv files is that the file with the combined node dumps contain more instances of the data but with fewer variables included which can be seen in figure 4.6.

The only difference in the naive Bayes implementation that uses several combined node dumps is the calculation of the measurements that measure the amount of identified transitions. When several files are used, several transitions need to be identified. This means that when four files are combined three transitions need to be correctly identified instead of one. One of the files does, as mentioned, not contain any transitions and will therefore not be inspected. This is done by evaluating the

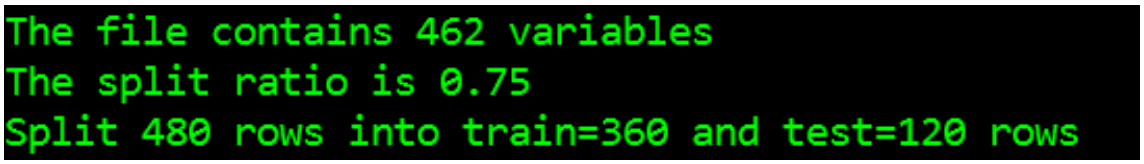

Figure 4.6: Illustrates how the combined node dumps are split into a training set and a test set.

amount of correctly classified transitions 200 times and calculating the mean of these values.

5

Results

The results of the implementation of naive Bayes is shown in section 5.1 and section 5.2. The results in section 5.1 describe how well the naive Bayes algorithm worked with a limited amount of data. The results in section 5.2 describe the results when the algorithm was fed with a bigger amount of data.

5.1 First iteration

This section describes the results of one run with the naive Bayes algorithm on the only node dump with a full set of variables. Figure 5.1 shows the resulting measurements from this run. The available data is split into four parts were three parts of the data, 180 instances, is used as a training set and one part is used as a test set, 60 instances. 157 of the total 240 instances are classified as bad and 83 instances are classified as good. The accuracy of the algorithm is calculated in two different ways both described in section 2.2. The first measurement in figure 5.1 shows the accuracy calculated by dividing the number of correct prediction of classes with the total amount of instances.

```
Split 240 rows into train=180 and test=60 rows
Accuracy: 92.00216450216446%
Total accuracy: 93.14935064935065%
Correlation coefficient: 0.8252772195891577
Percentage of identified transitions: 70.3225806451613 %
Percentage of transitions identified before they happened: 70.3225806451613 %
```

Figure 5.1: Results of one run with the Naive Bayes algorithm with one node dump.

As the classes of the variables in the test set are known the number of correct estimations can be counted and the accuracy with the test is approximately 92%. The second measurement shows the accuracy but with the false positives included which is about 93%, i.e. 93% of the data was classified correctly or classified as bad when it actually was good. This means that 7% of the data is classified as good, no overload happening, when actually it is bad, overload happening. The Matthews correlation coefficient lies at 0.8 which means that the predictions most likely could

not happen by chance. Any value above 0 constitutes as good and anything close to 1 as very good [20].

The amount of correctly classified transitions is approximately 70% which means that the only transition present in the file is incorrectly classified 30% of the time. The instance that precedes the overload is also classified as an overload 70% of the time. This means that in 70% of the runs the overload was identified an hour or more before it happens.

5.2 Second iteration

The results of a run with the naive Bayes algorithm with the full node dump and the second node dump is described in this section. Figure 5.3 shows the resulting measurements from this run. In this file the 480 instances are split into 360 instances used as the training set and 120 instances used as the test set and all of these instances are classified as bad. This file contains 462 variables which is a reduction of 1548 variables from the full file. The accuracy of the algorithm with these two files is 84.73%, which is roughly a drop of 7% compared to when the algorithm was run with one file. The total accuracy, i.e. the amount of correctly labeled classes and the amount of false positives, is approximately 86% which is also a drop of 7%.

```
Split 480 rows into train=360 and test=120 rows
Accuracy: 84.73749999999997%
Total accuracy: 85.98749999999998%
Correlation coefficient: 0.8086161522942495
Percentage of identified transitions: 86.0 %
Percentage of transitions identified before they happened: 86.0 %
```

Figure 5.2: Results of one run with the Naive Bayes algorithm with two node dumps.

The Matthews correlation coefficient is calculated to 0.81 which is a drop by 0.01 from the first run. The number of correctly identified transitions is 86% which is a raise of 16%. The amount of instances preceding the overload classified as overloads is also 86% which is also a 16% raise even though the file does not contain any more transitions. After this run the third node dump is added to the combined file which makes the total number of instances 753. These instances are split into 564 instances which make up the training set and 189 instances which make up the test set shown in figure 5.3. 273 of these instances comes from the third file were 33 of these instances are classified as bad and 240 are classified as good. The third node dump only contains 306 of the 462 variables the previous files had in common.

The accuracy of the naive Bayes algorithm with the three mentioned node dumps is 92.09% which is an increase of 7% from the run with two node dumps. The total accuracy is 92.10% which is fairly the same as the accuracy which only measures

```
Split 753 rows into train=564 and test=189 rows
Accuracy: 92.09259259259262%
Total accuracy: 92.09788359788362%
Correlation coefficient: 0.8486806576005302
Percentage of identified transitions: 100.0 %
Percentage of transitions identified before they happened: 66.16666666666667 %
```

Figure 5.3: Results of one run with the Naive Bayes algorithm with three node dumps.

the correctly classified instances. The correlation coefficient is calculated to be 0.825 which is a bit higher than the coefficient is in the run before. The amount of correctly identified transitions is 100% which is an increase of 14% from the run before. The instance prior to the overload is identified as an overload 66% of the time. The last node dump added to the file is the fourth node dump which contains 418 instances of which 333 are classified as bad and 85 are classified as good. This makes the total number of instances in the combined file 1171 instances long. The fourth node dump also has 306 variables in common with the other files as the third node dump had.

```
Split 1171 rows into train=878 and test=293 rows
Accuracy: 76.73720136518766%
Total accuracy: 88.8361774744027%
Correlation coefficient: 0.5833186148377812
Percentage of identified transitions: 88.16666666666667 %
Percentage of transitions identified before they happened: 54.833333333333336 %
```

Figure 5.4: Results of one run with the Naive Bayes algorithm with four node dumps.

The fourth run containing all of the node dumps combined has an accuracy of 76.74% which is the lowest score of all of the run and is a drop by 15% from the run with three files. The total accuracy is a bit higher, 88.84%, which is only a drop with 3% from the run before. The Matthews correlation coefficient is also the lowest so far, calculated to 0.55. The amount of identified transitions is 88% which is a decreased by 12%. The instance before the overload is classified as an overload 54.8% of the time. In table 5.1 a summary of all the runs with the naive Bayes algorithm can be found. The table includes the total number of combined instances as well as the combined number of good and bad instances. Besides that it also shows how many variables are used with the algorithm and the results of the accuracy measurements and the calculated Matthews correlation coefficient. The measurement of the total accuracy has been used to calculate the amount of false positives, i.e. how many instances are classified as good (0) when they should have been classified as bad (1).

	1 Log	2 Logs	3 Logs	4 Logs
Combined nbr of Instances	240	480	753	1171
Combined Bad Instances	157	397	430	763
Combined Good Instances	83	83	323	408
Nbr of Common Variables	2010	462	306	306
Accuracy	92.002%	84.737%	92.093%	76.737%
Total Accuracy	93.149%	85.987%	92.098%	88.836%
Amount of False Negatives	6.851%	14.013%	7.902%	11.164%
Matthews Correlation Coefficient	0.825	0.808	0.849	0.550
Identified Transitions	70.323%	86%	100%	88.167%
Pre-Identified Transitions	70.323%	86%	66.167%	54.833%

Table 5.1: A table of all different measurements of the runs with the naive Bayes algorithm.

6

Discussion

This chapter discusses the results obtained by using the naive Bayes algorithm with the node dumps produces by the SGSN-MME and also includes threats to validity and possible future work. The first section discusses the results of the research in relation to the research questions. The second section discusses the validity threats to the research and the third chapter describes the possible future work that could be performed.

6.1 Research Questions

The research that was carried out with three research questions in mind as the research was planned to include three steps. The first step was to identify an algorithm that could be used with the problem, the second step was to prepare the data and to implement the algorithm with that data and the third step was to evaluate the outcome. These steps and the results from them will be discussed in this section. The first research question was:

Which machine learning algorithm is suitable to use when predicting if a node in the cellular network is about to be overloaded?

This question was answered with the help of the literature review where different algorithms were evaluated to find an algorithm that worked with the PM data. One answer to this question is therefore naive Bayes as it was chosen as the classification algorithm to use with this problem. Naive Bayes was picked because it can consider many different variables at the same time and it also treats all variables equally which allows it to identify patterns that could seem insignificant to other algorithms. But even though naive Bayes was chosen many other algorithms could probably have worked just as well. When this question was answered it was possible to consider the answer to the next question, which was:

How can machine learning be used to identify the patterns in the PM data that causes CPU overloads in the SGSN-MME node?

As all machine learning algorithms work differently the suitable algorithm had to be chosen before the next question could be answered. Naive Bayes, for example, needs the standard deviations of the variables for it to be able to predict classes, which means that all variables that do not have a standard deviation, i.e. static variables, had to be removed. When naive Bayes was chosen the data could be prepared in the correct way for it to work with the algorithm. The answer to this question will therefore be that the data has to be prepared in a certain way for machine learning to work with it. The exact way the data was prepared can be seen in chapter 4, but it essentially included removing all static variables, classifying the data as good or bad, removing unnecessary information and making sure that all instances contain the same variables.

The second part of the answer of this question is the actual prototype that used naive Bayes to predict the classes, more close details on this can also be found in chapter 4. As seen in chapter 4 and 5 the prototype was successful in using naive Bayes to classify data instances from the PM data. Which leads up to the third question which is:

Is it suitable to use a classification algorithm to predict overloads in the cellular network?

This question can be answered by evaluating the measurements implemented with the first and second prototype of the naive Bayes algorithm. As seen in chapter 5 the results of running the naive Bayes algorithm with the available node dumps are relatively good. The accuracies all exceed 70% as was stated as the threshold value in chapter 3. The Matthews correlation coefficient is also relatively good as it always exceeds 0.5. The most successful runs was the first and third run were both runs exceeded an accuracy of 90% and the Matthews correlation coefficient was above 0.8. The second run had a bit inferior results in the accuracy department, but the Matthews correlation coefficient was still lied above 0.8 The worst run was the fourth run were exact accuracy lied below 80% and the accuracy including the false positives lies just below 90%. The Matthews correlation coefficient in the third run lies just above 0.5.

Looking at the numbers presented in the previous paragraph the algorithm has performed well in all areas. The values of the accuracy measurements and the Matthews correlation coefficient are all above the required thresholds. The performance of the algorithm is the worst when it is run with all four node dumps, but it still exceeds the threshold even in this run. The maximum amount of missed overloads is 14% which happens in the second run when only two files are used, which could be due to the fact that only instances classified as overloads are introduced in this run. Both the second and fourth run misses more overloads than wanted. The amount of missed overloads was preferably below 10%, but this problem might be amended gives that the file is optimized with more data. Overall the algorithm performs well in all runs, but it performs worse when the percentage of good instances is lowered.

The number of identified transitions varies but the pattern is that the amount of correctly identified transitions increases with the amount of data. In the combination were three logs are used the transitions are identified a 100% of the times which suggests that the teaching set the algorithm uses could be optimized further. This could create a version of the algorithm that identifies the large majority of the overloads before they occur. The instances that precede the overload are classified in the good class, class 0, but the measurement measures how many of these instances are classified as overloads. This measurement shows that many of the instances preceding the overload are identified as overloads, which means that it might be able to identify the overloads even earlier. If the instances preceding the overload were also classified as overloads it could be possible to identify them more than an hour ahead of time.

The amount of instances that are identified as overloads before the overload occurs decreases with the amount of training data. This seems logical as the algorithm better can learn when the overload is happening when more transitions exist in the training data. For the algorithm to both be better at identifying the overloads before they are happening and after they have happened more node dumps that contain transitions should be used. The most important measurement of them all is the amount of identified transitions because this is the key classification task for the algorithm. If the algorithm is not able to identify the transitions it is not able to identify the overloads. It is therefore a positive indication that the amount of identified transitions is as high as 100% in some of the cases as it indicates that the algorithm can be optimized with more training data to produce good results.

6.2 Validity Threats

An identified threat to this research is that the number of possible real world node dumps in the database that could be used with the implementation in this thesis was relatively limited. In the beginning of the research a total of nine node dumps were identified as being likely candidates to use with the naive Bayes algorithm. The amount of available real world data was limited from the start but these nine files were identified and chosen as they had recorded overloads in the node of interest. At a closer examination it was found that many of the files did not contain the amount of variables they were supposed to contain. Only one node dump with a full set of variables was found and it was therefore decided that one prototype that used the full set of variables and one prototype that only used the common variables in all node dumps was to be developed.

Another problem was that most of the node dumps contained a very small amount of good instances or no good instances at all. The low amount of good values is not representative for the real world events as good events stands for about 99% of the real world events and the bad events are exceptions not the rule. It is therefore important that a relatively large amount of good instances is present in the files for the algorithm to be valid. A third problem with a number of the node dumps

was that they contained variables without headers which made it problematic to combine them with other files. These node dumps were therefore excluded from the research which decreased the number of node dumps to only four.

In the four chosen files only 33% of the instances were labeled as good and 65% were labeled as bad. Another problem is that all used node dumps together only contain three transition instances and one pre-transition instance. The consequence of this is that the algorithm had to work with a limited amount of data which might result in a model that only works with specific problems. An overload could happen because of a number of different reasons and because of many different abnormal signaling patterns. If all of these events are not represented in the node dumps used to train the algorithm the algorithm can not make a model of such events. This could therefore lead to these specific problems being missed in real world use of the algorithm and therefore the generalizability of the method could be limited.

Another threat to the research is the limited amount of variables. The only full node dump contains all of the variables but the other node dumps contain less than half of the total amount of variables. This is a problem since the node dumps chosen should by specification contain all of the variables but many of them have not been recorded. This could mean that the cause of the overload lies in one, or several, of the non-recorded variables which will be missed by the algorithm.

6.3 Future Work

If naive Bayes is chosen as the machine learning technique that will be used to identify the overloads in the SGSN-MME node it needs to be validated with more real world data as a first step of action. More real world data that includes both good and bad events is needed to fully optimize the algorithm. This could make the algorithm as robust as it needs to be for it to be used to identify overloads in real time with the node. Then if the accuracy of the optimized algorithm is as good or better than it is in this research possible preventative software mechanisms should be evaluated so that the algorithm could be integrated in the current system. As there is an alarm system already in place, the one that identifies the overloads when they happen, the algorithm could be integrated with that alarm system to warn technicians about possible coming overloads. If the technique is successful it could be extended to other nodes in the network.

7

Conclusion

This thesis has aimed to evaluate the possibilities of using a classification machine learning technique with numerical data from a specific node in a cellular network. Several machine learning algorithms were considered during the process as many classification algorithms could be used to solve the problem, but naive Bayes was ultimately chosen as the algorithm of choice. The reason for this was that the multitude of signaling variables that the PM data contains are all seen as equal by the naive Bayes algorithm which allows them to be combined to create bigger effects. The proof of concept was then used to evaluate if the chosen technique could be used to predict overloads in the component of interest in the cellular network. The proof of concept resulted in several measurements that were used to evaluate the prototype.

The research showed that the proposed machine learning technique worked well with the problem at hand since the algorithm was able to identify the majority of the overloads it was presented with. The Matthews correlation coefficient is high enough to indicate that the algorithm did not predict the classes that it did by chance. The study also shows that the accuracy of the algorithm varies depending on the amount and type of data that is used as input. It illustrates examples were a greater amount of data does not directly imply a greater accuracy as one might have expected. It also seems as though the accuracy increases when the both classes are represented equally in the data.

More research on the topic needs to be performed but the research seems to indicate that naive Bayes can be used to predict overloads in hardware components with the use of numeric data. This has to be tested with, for example, other nodes in the cellular network or other hardware components. The research gives a concrete example of how a classification algorithm can be used to predict extreme events. This example should be applicable to other similar situations were a component only can handle a limited amount of signaling. The classification can then be used to take appropriate action to minimize the problems that could occur if the component is overflowed with signals.

Limitations to the study includes the minimal amount of available data which makes it difficult to draw conclusions about generalizability. Also the solution has only been

applied to one specific node in the cellular network and it is not certain that the solution would work as well with other nodes. As a conclusion it can be said that the proposed machine learning technique worked well with the problem at hand even though it needs to be further evaluated to conclude if it can be generalized to other contexts.

Bibliography

[1] E. Svensson *Optimized Routing within an Ericsson Node - Routing procedures in an Ericsson SGSN.* M.S. thesis, Dept. Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2011.

[2] M.N. Murty and V.S. Devi. *Introduction to Pattern Recognition and Machine Learning.* (2015). [Online]. 5. Available at: http://app.knovel.com/hotlink/toc/id:kpIPRML003-/introduction-pattern/introduction-pattern [January 18, 2017].

[3] O. Kramer. *Machine Learning for Evolution Strategies.* (2016). [Online]. 20. Available at: http://link.springer.com.proxy.lib.chalmers.se/book-/10.1007%2F978-3-319-33383-0 [January 19, 2017].

[4] M. Paluszek and S. Thomas. *Matlab Machine Learning.* (2017). [Online]. Available at: http://download.springer.com.proxy.lib.chalmers.se/static/pdf/490/ [January 19, 2017].

[5] T. C. Silva and L. Zhao. *Machine Learning in Complex Networks.* (2016, November). [Online]. Available at: http://link.springer.com.proxy.lib.chalmers.se/book/10.-1007%2F978-3-319-17290-3 [January 18, 2017].

[6] Y. Anzai. *Pattern Recognition and Machine Learning.* (1992). [Online]. Available at: http://www.sciencedirect.com.proxy.lib.chalmers.se/science/article-/pii/B9780080513638500016 [January 19, 2017].

[7] C. Ebert and P. Louridas. *Machine Learning.* (2016, August 24). IEEE Software. [Online]. 33(5), pp. 110-115. Available at: http://ieeexplore.ieee.org.proxy.lib.chalmers.-se/document/7548905/ [January 19, 2017].

[8] A.R. Hevner, S-T. March, J. Park and S. Ram. *Design Science in Information Systems Research.* (2004). [Online]. Available at:

http://wise.vub.ac.be/thesis_info/design_science.pdf [January 19, 2017].

[9] M.A. Hall. *Correlation-based Feature Selection for Machine Learning.* M.S. thesis, Dept. Computer Science, The University of Waikato, Hamilton, New Zealand, 1999.

[10] J. Sui. *Understanding and fighting bullying with machine learning.* Ph.D. thesis, Debt. Computer Science, University of Wisconsin-Madison, Madison, USA, 2015.

[11] S.R. Modarres Najfabadi. *Predictions of Stock Markets Indices using Machine Learning.* M.S thesis, Debt. Computer Science, McGill University, Montreal, Canada, 2009.

[12] B. Lantz. *Machine Learning with R.* (2013). [Online]. Available at: https://app.knovel.com/web/toc.v/cid:kpMLR0000G/viewerType:toc/-root_slug:machine-learning-with/url_slug:kt00U5RAX2 [February 3, 2017].

[13] V. Vaishnavi and W. Kuechler. *Design science research in information systems.* (2004). [Online]. Available at: http://desrist.org/desrist [March 27, 2017].

[14] S. Theodoridis. *Machine Learning, A Bayesian and Optimization Perspective.* (2015). [Online]. Available at: http://www.sciencedirect.com.proxy.lib.chalmers.se/science/-article/pii/B9780128015223099936 [April 24, 2017].

[15] A. R. Webb, K. D. Copsey and G. Cawley. *Statistical Pattern Recognition.* (2011, September 15). [Online]. Available at: http://ebookcentral.proquest.com/lib/chalmers/-reader.action?docID=819173 [April 24, 2017].

[16] C. Sammut and G. Webb. *Encyclopedia of Machine Learning.* (2010). [Online]. Available at: https://link-springer-com.proxy.lib.chalmers.se/referencework/-10.1007%2F978-0-387-30164-8 [April 26, 2017].

[17] L. Yinhui et al. *An efficient intrusion detection system based on support vector machines and gradually feature removal method.* (2012, July 22). Expert Systems with Applications. 39(1). [Online]. Available at: http://www.sciencedirect.com.proxy.lib.chalmers.se/science/-article/pii/S095741741100994 88 [April 26, 2017].

[18] M. Shepperd. *How Do I Know Whether to Trust a Research Result?.* (2015,

February 4). IEEE Software. 32(1), pp. 106 - 109. [Online]. Available at: http://ieeexplore.ieee.org.proxy.lib.chalmers.se/document/7030205/ [April 26, 2017].

[19] G. Jurman and C. Furlanello. *A unifying view for performance measures in multi-class prediction.* (2010, January 17). [Online]. Available at: https://arxiv.org/pdf/1008.2908.pdf [April 27, 2017].

[20] J. Gorodkin. *Comparing two K-category assignments by a K-category correlation coefficient.* (2004, December). Computational Biology and Chemistry. 28(5-6), pp. 367–374. [Online]. Available at: http://www.sciencedirect.com.proxy.lib.chalmers.se/science/-article/pii/S1476927104000799 [April 27, 2017].

[21] P. Baldi et al. *Assessing the accuracy of prediction algorithms for classification: an overview.* (2000, February 23). Bioinformatics Review. 16(5), pp. 412–424. [Online]. Available at: http://web.cs.iastate.edu/ jtian/cs573/Papers/Baldi-etal--Bioinformatics-2000.pdf [April 27, 2017].

[22] H. Zhang et al. *Feature selection for optimizing traffic classification.* (2012, July). Computer Communications. 35(12), pp. 1457–1471. [Online]. Available at: http://www.sciencedirect.com/science/article/pii/-S0140366412001259 [May 15, 2017].

[23] Y. Elovici et al. *Applying Machine Learning Techniques for Detection of Malicious Code in Network Traffic.* (2007). Annual Conference on Artificial Intelligence. 4667, pp. 44-50. [Online]. Available at: https://link.springer.com/chapter/10.1007/-978-3-540-74565-5$_5$[$May 15, 2017$].

[24] M. Soysala and E.G. Schmidt. *Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison.* (2010, June). Performance Evaluation. 67(6), pp. 451–467. [Online]. Available at: http://www.sciencedirect.com/science/article/pii/-S0166531610000027 [May 15, 2017].

[25] D.M. Bui et al. *Gaussian process for predicting CPU utilization and its application to energy efficiency.* (2015, July 2). Applied Intelligence. 43(4), pp. 874–891. [Online]. Available at: https://link-springer-com.proxy.lib.chalmers.se/article/-10.1007%2Fs10489-015-0688-4 [May 16, 2017].

[26] D. Yang et al. *A pattern fusion model for multi-step-ahead CPU load prediction.* (2013, May). Journal of Systems and Software. 86(5), pp. 1257–1266. [Online].

Available at:
https://link-springer-com.proxy.lib.chalmers.se/article/-10.1007%2Fs10489-015-0688-4 [May 16, 2017].

[27] D. Minarolli, A. Mazrekaj and B. Freisleben *Tackling uncertainty in long-term predictions for host overload and underload detection in cloud computing.* (2017, February). Journal of Cloud Computing: Advances, Systems and Applications. 6(1), pp. 1–18. [Online]. Available at:
https://link-springer-com.proxy.lib.chalmers.se/article/-10.1007%2Fs10489-015-0688-4 [May 16, 2017].

[28] J. Huang, J. Lu and C.X. Ling. *Comparing naive Bayes, decision trees, and SVM with AUC and accuracy.* (2003, November 22). Third IEEE International Conference on Data Mining. [Online]. Available at:
http://ieeexplore.ieee.org.proxy.lib.chalmers.se/document/-1250975/ [May 16, 2017].

[29] Y. Qian et al. *Comparing Machine Learning Classifiers for Object-Based Land Cover Classification Using Very High Resolution Imagery.* (2014, December 24). [Online]. Available at:
http://www.mdpi.com/2072-4292/7/1/153/htm [May 16, 2017].