A Genetic Feature Selection Algorithm for Anomaly Classification in Mobile Networks

Márton Kajó and Szabolcs Nováczki
Nokia Networks, Budapest, Hungary
{marton.kajo, szabolcs.novaczki}@nokia.com

***Abstract*— The operators of big mobile networks rely on operation support systems (OSS) to help in the setup and management of these expansive networks. Automating some functions in the OSS could reduce operation costs and increase efficiency. One such function is the diagnosis of abnormal behavior in the network's systems stemming from incorrect settings, insufficient network capabilities, software or hardware malfunction. This paper presents a system that can detect and classify such anomalous behavior with the data provided by logging functions in currently existing OSS. We show that it is possible to detect complex user defined states in the network from seemingly not closely related indicators with the use of classification algorithms. The presented system implements a wrapping feature selection algorithm that can select the most important performance indicators for the classification task. The system was tested with four different classifiers, the KNN (K-Nearest Neighbors), CART (Classification And Regression Trees), SVM (Support Vector Machine) algorithms and a Neural Network. We show in this paper how feasible the concept is, and how well these classifiers performed on data provided by a mobile network.**

Keywords: OSS, Genetic Algorithm, Feature Selection, KNN, CART, SVM, Artificial Neural Network

I. Introduction

Operation support systems provide diverse information about the network in the form of Key Performance Indicators (KPI) that can be used for simple fault detection. However, more complex network states that do not necessarily represent erroneous behavior are hard to recognize from a single indicator. State recognition using more than one indicator requires the design of multi-dimensional classification rules, a task that can be carried out autonomously by classification algorithms. The logical step is to use these algorithms to process the data provided by the OSS to extract information that identifies various network states. This information could later be used for optimization or reconfiguration of the network. The goal of our research was to see whether such an automatically generated multi-dimensional classification system can be used for anomalous state diagnosis in mobile networks.

Classification algorithms have seen a great surge in utilization in the recent years, as more and more applications implement voice or image recognition. These algorithms are now used in different areas, such as medical diagnostic systems or big data analysis. While the performance of these algorithms is constantly improving as more effective versions are developed, computational constraints still plague the user. As a general rule, classification algorithms have a hard time functioning in high-dimensional spaces, a phenomenon commonly referred to as the curse of dimensionality [1]. For this reason, it is usually beneficial to use a dimensionality reduction technique, such as feature selection, when using a classification algorithm on high-dimensional data. Since the OSS provides information from all aspects of the network, it is highly likely that some, or most of the information is irrelevant to the current use case. The aim of feature selection is to reduce the dimensionality of the feature space by eliminating features (dimensions) that contain irrelevant or redundant information. By using only the most important features, the classification algorithms can function faster and create simpler, more precise classification rules.

Selecting the optimal subset of dimensions for classification from a high-dimensional space is an NP-complete problem [2], and is unsolvable by exhaustive search. Such computationally intractable problems can be solved with the use of genetic algorithms. These algorithms mimic an evolutionary process in their search, and can find a suitable solution in less than polynomial time [3]. We present a genetic algorithm in this paper that, combined with a classification algorithm, can select the most important KPIs to detect a given network state.

Previous research in mobile network anomaly detection conducted by Nokia [4] used a single class SVM as classifier, where the main goal was the identification of cell-service performance degradation. Similar work consists of [5], where a Naive Bayesian classifier diagnosed dropped calls, and [6], where a neural network combined with competitive learning was used for fault detection. In all of these, the KPIs given to the detection systems were hand picked by the researchers. The system presented in this paper can automatically select the most important KPIs for the given classification task, which makes it easy to adapt to different diagnostics tasks, and does not require expert knowledge of the network.

Wrapping feature selection using a genetic algorithm is not a novel idea. It was first proposed in [7], which used a neural network as a classifier. [8] compared this wrapper approach to some well-established filter-type subset selection algorithms. [9] and [10] used a support vector machine as a classification algorithm. The primary goal of these researches

was to improve the classification accuracy by using only the most important features. We were also interested in whether the selected features made sense from a network engineering perspective and could be strongly linked to the measured network states.

To give a better overview, following are the basic steps to use the presented system:

1) Create a training dataset by labeling previously measured data provided by the OSS, or as in our case, measure a test network set up for the different states.
2) Run the feature selection process with the classification algorithm of choice.
3) Use the selected features to train the final classifier.
4) Diagnose the network in real time with the trained classifier.

The rest of the paper is organized as follows: First, we give a brief overview about classification algorithms in general, and the four classification algorithms tested to work with the system (II). After this a short introduction to Genetic Algorithms (GA) follows, with a more detailed explanation of how the feature selector incorporates the classification algorithms (III). Finally, we show the results of testing this feature selector on data extracted from a mobile network (IV), and close the paper with a conclusion (V).

II. Classification

A part of machine learning, classification algorithms realize learning systems that create classification rules by processing a training dataset. Classification algorithms fall in the supervised learning category, since they require the training examples to be labeled according to which class they belong. The created classification rules can then be used to predict the class of unlabeled observations. A class can be any quantifiable property of the observations, such as a color, or in this case, the state of the network. An everyday example of a classification algorithms is the spam filtering mechanism in e-mail software. Here the user manually labels the incoming e-mails as unwanted, from which the system learns, and can later filter automatically.

When forming the classification rules, the algorithms sometimes create rules that fit the training data too precisely. This is undesirable, as the training observations usually contain noisy data that do not represent the distribution of the classes well. This phenomenon is called overfitting. A good classification algorithm is expected to have some degree of generalization when forming the classification rules. For this reason, most classification algorithms have a set of parameters that control the generalization during training.

To figure out the correct setup of the algorithms' parameters, one can use some form of validation in order to avoid overfitting the training data. Validation techniques are designed to test the performance of the classifier using only the original training data by splitting the data into a test and a training set. The most commonly used validation techniques are random sampling and k-fold cross validation. Being able to measure the performance of the classifier allows the user to find the optimal set of classifier parameters using a grid search.

The wrapper nature of the feature selector allows it to work with any type of classification algorithm. Having plenty of computing power at our disposal, we opted to try out a number of the most commonly used classification algorithms. Our choice fell on four algorithms: the K-Nearest Neighbors (KNN) algorithm, the Classification And Regression Trees (CART) algorithm, a Support Vector Machine (SVM) classifier, and an Artificial Neural Network (ANN). All of these realize different methods to form the classification rules, and thus have different computational costs and overall classification accuracy.

A. K-Nearest Neighbors

The KNN is algorithmically the simplest classifier of the four. The KNN algorithm classifies an unlabeled observation by searching for the closest K number of neighbors of the data point in the training set. The classification is then computed by a majority vote, the assigned class is that which has the most representatives amongst the neighbors. The number of neighbors taken into account during the voting greatly influences the classifier's tendency to overfit [11]. The correct K value can be set by an exhaustive grid search. The KNN algorithm doesn't have a training phase, but merely stores the training data, and only requires the parameter K to be set. This makes training and optimizing the classifier really fast in the context of the feature selection algorithm.

B. Classification And Regression Trees

The CART algorithm is a decision tree based method [12], incorporating both regression and classification functions in a single algorithm. The algorithm creates a decision tree during its training phase, where every node represents a binary (yes/no) question about one of the features in the search space. The tree is built by recursively splitting every subspace along one of the features, starting with the whole search space. CART uses the Gini index as a metric to decide along which feature the split should be. In order to avoid overfitting the training data, the tree is pruned back after the building process with a pruning algorithm. The pruning algorithm decides which nodes to cut by comparing the lost classification accuracy to the reduction in the tree's complexity. The threshold ratio C is the parameter that controls the pruning, and can be set by a grid search.

C. Support Vector Machine

The SVM classifier [13] finds the optimal separating hyperplane between classes in a transformed, high dimensional

feature space. It works with a soft margin hyperplane that allows for points in the separating margin and wrongly classified points in favor of a better generalization. The fitting of the soft margin hyperplane can be controlled by a cost parameter. The optimal separating hyperplane can be found in the transformed space by knowing the inner products of vectors in that space. These can be calculated using a kernel function in the original feature space, which allows the algorithm to avoid the calculation of the whole transformation. The kernel function we used was the frequently utilized radial basis kernel function, which requires the kernel's gamma parameter to be set. This, and the cost parameter need to be set by the user carefully, since both can influence the classifier's tendency to overfit.

D. Single Hidden Layer Neural Network

Artificial neural networks consist of interconnected nodes called neurons. These neurons take multiple inputs, sum them, and create the node's output with this sum through a sigmoid activation function. The nodes are organized in layers, with connections only running between nodes in different layers. The neural network used in this research is a single hidden layer network [11] containing three layers: an input layer, a hidden layer and an output layer. The number of nodes in the input and output layers are determined by the number of features and classes used in the classification task. The number of nodes in the hidden layer however, are up to the user to set, and can cause the network to overfit. The learning phase consists of setting the weights of the connections so that the classification error is minimal on the training set. This is done through an iterative advanced back propagation algorithm. The system also has weight decay at each iteration to evade overfitting. The amount of weight decay and the number of hidden layer nodes all need to be set by the user or through exhaustive grid search. Combining this with the long iterative training phase makes for a really slow to train, albeit extremely accurate classifier.

III. Feature selection using a genetic algorithm

Genetic algorithms (GA) are heuristic search algorithms. These algorithms use historical data to narrow the search on the best performing areas of the search space through an iterative process that resembles natural selection. Genetic algorithms work on a group of potential solutions simultaneously called a generation, where each individual is represented by a numerical vector called a chromosome. With each iteration, a genetic algorithm creates a new batch of chromosomes from the previous generation's chromosomes through the genetic operators of selection, crossover and mutation. For this to work, each chromosome needs an assigned fitness score that represents the goodness of the corresponding solution. The fitness score is calculated via a fitness function. The genetic algorithm's tasks are the following at each iteration:

1) Calculate the **fitness** score for each chromosome.
2) **Select** the parent chromosomes according to their fitness scores.
3) Create new chromosomes through **crossover** by combining the selected parent chromosomes.
4) **Mutate** some of the new chromosomes.

Genetic algorithms can be easily modified to solve a wide range of problems by changing the fitness function and the way a chromosome represents a solution. Originally these algorithms were invented for optimal subset selection, with the chromosomes being binary vectors [14]. There now exist modified versions of genetic operators that can solve permutation-like problems or can work with real valued chromosomes. Since our goal was to find an optimal feature set, the original binary chromosome representation worked just fine. This way, each of the chromosome's bits represent whether or not the assigned feature is included in the set. This also made the design process easier, since binary chromosomes have many well researched genetic operators to choose from.

There are many examples of feature selection systems using genetic algorithms that implement a filter-like selection, where the fitness function only uses the training data to calculate the fitness value, but not the actual classifier [15]. The approach presented in this paper is a wrapper approach in the sense that it incorporates the classification algorithm in the fitness function. With this the chosen features are tailored to the specific classification algorithm that was used in the search.

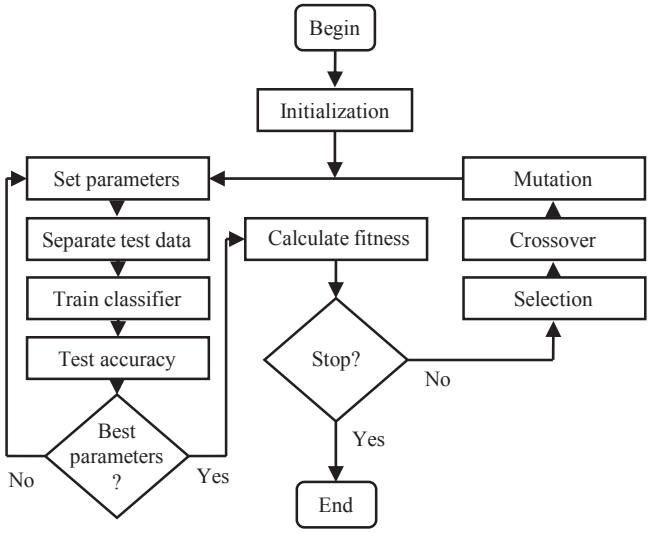

Fig. 1. The full feature selection algorithm

A. Fitness function and classifier setup

The fitness function's goal is to incorporate the different optimization criteria, and give a numerical value to the goodness of the chromosome according to these criteria. Our aim was to use as few features as possible while still maintaining a

good classification accuracy. We chose the linear combination of these two variables as our fitness function:

$$fitness(x) = accuracy(x) \times 100 - n_{\text{features}}(x), \quad (1)$$

where x is a chromosome, *accuracy(x)* is the classification accuracy with the chromsosme, and $n_{features}(x)$ is the number of features (KPIs) present in the chromosome. The classification accuracy is measured on a classifier that was trained with the features represented by the chromosome. The tradeoff between used features and classification accuracy is represented in the ratio between the two variables. As it stands here, 1% gain in classification accuracy justifies the addition of a new feature. To determine the accuracy of the classifier, uniform random sampling was used where 10% of the training data was selected as test data. The classifier was trained on the training data excluding the test data, and then tested with the remaining observations. The random sampling and training was repeated three times, and the final value was the average of accuracies across the repetitions.

To properly measure the efficiency of a classification algorithm, one has to set the different parameters of the classifiers. This was done with a grid search on the classifiers with faster training phases (KNN and CART) by trying out every possible combinations of pre-set values. Grid searching proved to be unfeasible on the algorithms requiring longer training times (SVM and ANN), since the global runtime would have been over a year with our equipment. Instead of grid searching, these algorithms used unoptimized pre-set parameters that leaned towards overfitting. This resulted in selected features that separated the classes by the biggest distance and with as less noise as it was possible, which was the main goal of feature selection all along. The downside is that the unoptimized classifiers theoretically allow the existence of better feature sets, that have worse classification accuracy with these parameters, than the ones the feature selector found.

B. Genetic operators

Selection was done with a linear-ranking scheme. Here the chromosomes are ranked according to their fitness score, and are chosen as parents proportionally to their rank. Linear-rank selection is a robust selection scheme that is insensitive to big differences in the fitness scores [16]. Additionally to this, we also used elitism with a single individual. This way the chromosome with the best fitness score gets carried over to the next generation without any change to it. This ensures that the best solution found so far cannot get lost through unfortunate crossover or mutation.

Crossover was done with a uniform crossover operator, where each gene has the same chance of coming from either parent. Uniform crossover seems to have better convergence when having smaller populations [17].

The chance of mutation occurring in a chromosome was 10%. If a chromosome was chosen to mutate, roughly 10% of its genes were flipped. This is a moderately high chance of mutation that encourages exploration of the search space and avoids premature convergence. Mutation also serves to dislodge the search if it's stuck at a local maximum.

C. Initialization and stopping

To create the chromosomes in the first generation, a random uniform distribution was used. Due to computing constraints in the classifiers, a maximum of 30 features were distributed in each chromosome. To combat the sparseness of the chromosomes, each generation was made up of 50 individuals, which together contained an average of 215.85 out of the 216 different KPIs, thoroughly covering the whole search space.

The genetic algorithm stopped if the best chromosome's fitness score didn't improve in the last 250 iterations, or if it reached the maximum of a 1000 iterations. This stop criteria was very generous, as early testing showed that convergence was achieved in roughly 250 iterations, and no GA improved its best score above 500 iterations.

IV. Results

To generate the training data, an LTE network was used comprising of a single cell, a base station and a gateway (Fig. 2). Multiple user equipment (UE) simulated normal user behavior by downloading files from a remote server. The throughput demand was varied throughout the tests by setting up different time intervals that the UEs waited between each file download. The OSS system connected to this network was Nokia's own OSS solution that collected information from the base station. To generate different network states, a traffic shaper was used at the gateway that comprised of a throughput limiter and a buffer before it. The shaper only tampered with data packets, but not control packets, and only in the downlink direction. The four states the network was measured in were the following:

- **(N) Normal**: All parameters of the transport network were set to allow for virtually unlimited throughput. The throughput limiting bottleneck was the LTE radio interface in this case, where a 15dB attenuation was present, which limited the overall throughput to about 30Mb/s.
- **(L) Long buffer**: In this case the shaper limited the passing data flow to 1Mb/s, while the buffer in the shaper was set to 1MB, which artificially delayed the passing packets up to 8 seconds. This delay and the throughput limit put a serious strain on the TCP flow control, resulting in a highly reduced overall throughput.
- **(S) Short buffer**: In this case the shaper limited the throughput to 20Mb/s, and the connected buffer was set to 28KB. While this did not result in a great throughput reduction, the small buffer frequently caused incoming packets to be tail dropped, further reducing the connections performance.

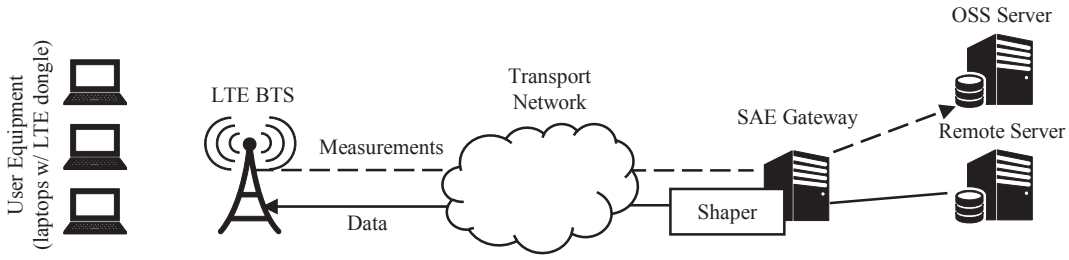

Fig. 2. The topology of the test network

- **(D) Discard**: Here the input buffer did not limit the throughput, but randomly discarded 5% of the incoming packets. As in the short buffer case, this also reduced the connections performance by forcing them to resend packets.

These four states formed the classes that the classifier algorithms had to distinguish. Each of the four states made up 1/4 of the total of 850 training observations. The 216 KPIs logged in the base station were averaged across 15 minute time intervals. The shaper was deliberately set up in the gateway, so that the collected information and the cause of the states were in separate network components. The classes are hard to distinguish solely on the achieved throughput, since the generated throughput demand varied during all the measurements. For reference a CART classifier properly optimized and trained on a single KPI corresponding to the logged throughput values could only achieve 65% classification accuracy on the training dataset.

The feature selector was written using the R[1] statistical computing language. The R language is widely used by statisticians and data miners, with available packages for most of the common algorithms and functions. For the KNN classifier, the *CLASS*[2] built in package was used, the CART algorithm came from Brian Ripley's *rpart*[3] package, the *e1071*[4] package implemented the SVM classifier, the K-fold cross validation and random sampling methods, and finally the neural network came from the *nnet*[5] package. The genetic algorithm used the framework from the *GA*[6] package. All of these packages are open source, which made putting together the feature selector a simple task.

A. Comparison of the classifiers in the feature selector

The feature selector was run 100 times with each of the classification algorithms. Table I shows the statistical values of the runs with optimized classifiers (KNN and CART). The biggest difference was in the time it took to run the algorithms; using a state of the art Intel Xeon processor with no parallelization, all the feature selectors using the KNN finished in an hour, whilst with the CART algorithm it took almost a week. This big difference is caused partly by the more possible values for the C parameter the CART algorithm was tested on than the K values for the KNN algorithm, and partly by the time required to build the CART classification tree. Though CART is faster during testing when evaluating unlabeled observations, the extra training time still makes this classifier an order of magnitude slower here than the KNN.

TABLE I
STATISTICAL INFORMATION OF USING THE FEATURE SELECTOR WITH THE OPTIMIZED KNN AND CART CLASSIFIERS

	KNN				CART			
	Min	Avg	Max	Std. dev.	Min	Avg	Max	Std. dev.
Runtime	0^h15^m	0^h30^m	1^h2^m	0^h8^m	48^h22^m	84^h55^m	166^h56^m	26^h46^m
Iterations	268	380.19	761	100.75	268	393	742	118.80
Accuracy	98.82	99.64	100	0.2678	98.59	99.54	99.88	0.2301
$n_{features}$	2	2.47	3	0.5016	2	2.35	3	0.4793

TABLE II
STATISTICAL INFORMATION OF USING THE FEATURE SELECTOR WITH THE UNOPTIMIZED SVM AND NEURAL NETWORK CLASSIFIERS

	SVM				ANN			
	Min	Avg	Max	Std. dev.	Min	Avg	Max	Std. dev.
Runtime	5^h56^m	13^h1^m	23^h38^m	4^h42^m	21^h11^m	42^h54^m	94^h41^m	16^h28^m
Iterations	270	500.75	1000	186.36	268	325.38	582	83.97
Accuracy	98.35	99.62	100	0.3541	99.05	99.97	100	0.1314
$n_{features}$	3	3.74	5	0.5794	2	2.11	3	0.3144

Table II shows the statistical values of the runs using unoptimized classifiers (SVM and ANN). Since these algorithms required multiple parameters to be set, the runtime reduction by not finding the correct combinations via grid search was considerable. Running the feature selector with the ANN classifier and optimization enabled would have taken an estimated 400 days with our hardware. The outlier of the two is the SVM algorithm, which preferred to use more KPIs than all the the other classifiers. Since the SVM works in a high-dimensional space through the kernel function anyway, it can use more features in the original feature space without

[1]https://www.r-project.org/
[2]https://cran.r-project.org/web/packages/class/
[3]https://cran.r-project.org/web/packages/rpart/
[4]https://cran.r-project.org/web/packages/e1071/
[5]https://cran.r-project.org/web/packages/nnet/
[6]https://cran.r-project.org/web/packages/GA/

suffering from dimensionality. While the capability does not necessarily mean preference, with our measurement data the SVM could only achieve a good classification accuracy by using more KPIs than the other algorithms. This also meant that the SVM classifier had the overall lowest fitness score on average.

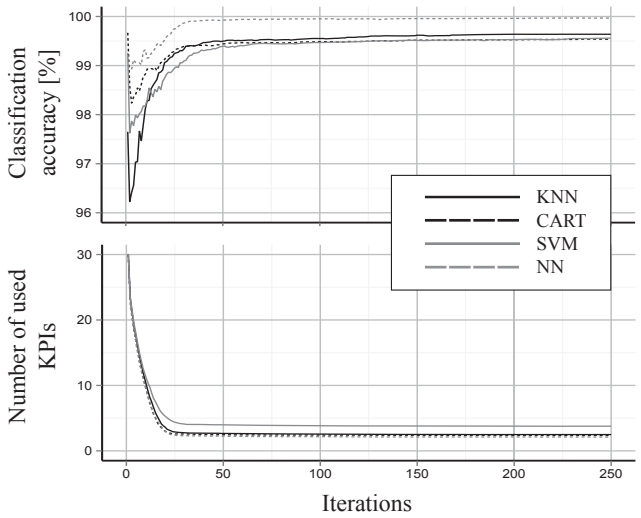

Fig. 3. The first 250 iterations of the genetic algorithm. Shown here is the best candidate's classification accuracy (upper), and the number of KPIs used in it (lower). The values are averaged across the 100 individual runs for all the classification algorithms.

The SVM and CART algorithms were capable of handling a larger number of dimensions before invoking the curse of dimensionality. This effect can be observed on the first few iterations (Fig. 3 upper), where these algorithms have better classification accuracy compared to the KNN or ANN classifiers. Unfortunately this advantage does not matter that much in the outcome, since after the first few iterations the classifiers only had to work on a couple of KPIs (Fig. 3 lower). All of the classifiers were able to achieve a close to perfect classification accuracy due to the training data not being very noisy. For one, this probably stems from the test environment not representing a real network perfectly, and not containing as much noisy measurement generating behavior, mostly because the radio channels were not disturbed by time-varying fading effects. With this in mind, we still believe that compared to other classification tasks in medicine or image recognition, measurement data from mobile networks does not contain as much noise because of the strong connections between the network's mechanisms, and can be classified with a simpler classifier with good accuracy. Hence, the KNN algorithm was perfectly sufficient in the context of the feature selector and performed much faster than the other algorithms.

B. KPIs chosen by the feature selector

The results provided by the KNN algorithm were in line with our expectations, with the classifiers not overly preferring a single KPI. Out of the 216 KPIs, 58 different ones were chosen at least once, with the most frequent KPI being represented in 28% of the outcomes. This same KPI was also chosen the most times by the SVM and ANN classifiers, which makes us believe that it indeed contains valuable information regarding the different network states. The distribution of the chosen KPIs can be seen on the right side of figure 4. The KPIs referenced in this paper mean the following:

- KPI_1 is proportional to the amount of retransmitted packages.
- KPI_2 is proportional to the amount of Transmit Time Intervals (TTI) where at least one user was active on the radio interface.

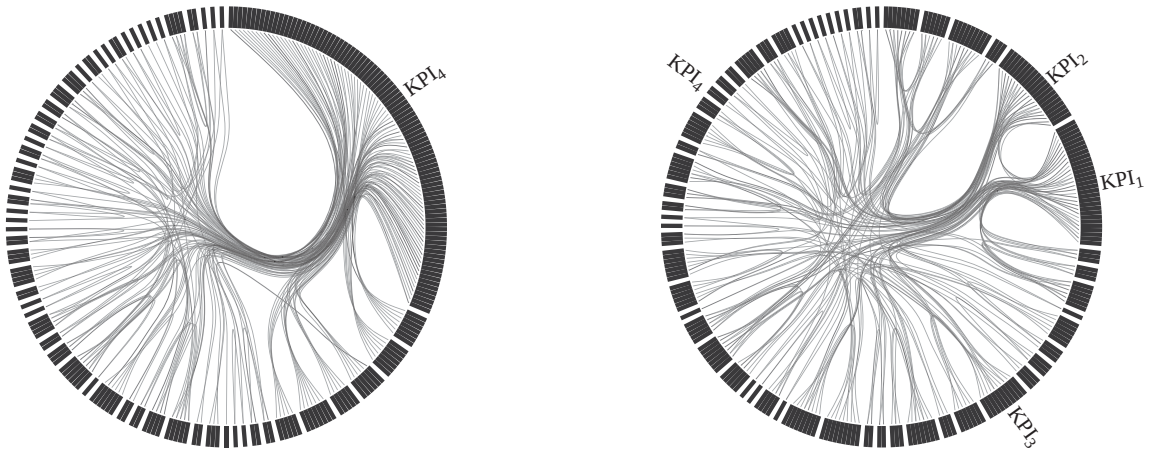

Fig. 4. Bundling graphs of the KPI sets chosen by the feature selector using the KNN (right) and CART (left) classifiers. A rectangle represents one occurence of a KPI, with connecting lines running to other KPIs that were present in that same set. The same KPIs are grouped together on the edge, the different KPI groups are separated by larger spaces.

- KPI_3 is proportional to the total amount of transmitted packages.
- KPI_4 is proportional to the mean length of the RLC (Radio Link Control) layer's buffer queue.

It is important to note that these KPIs were usually not chosen together, with the classifier instead preferring one of the other, lesser represented KPIs in combination with these.

The CART classifier proved to be problematic by including the same KPI (KPI_4) in 98% of its indicator sets (Fig. 4 left). This could mean that KPI_4 can separate the classes really well, but the same KPI was underrepresented in the other algorithm's results (for example using the KNN algorithm), which makes this suspicious. Further inspection of the classification rules generated by the CART algorithm revealed that this KPI was special because it allowed for a relatively precise split of the classes with separators perpendicular to this feature, which was preferred by the CART algorithm. None of the other features' use created a distribution where the optimal separating planes were so parallel or perpendicular to the features. This preference unfortunately masks the real information value of the indicators, but provides a valuable lesson; each classification algorithm can prefer different KPIs depending on its mechanics, and only the KPI sets acquired with the same classification algorithm are optimized for that specific classifier.

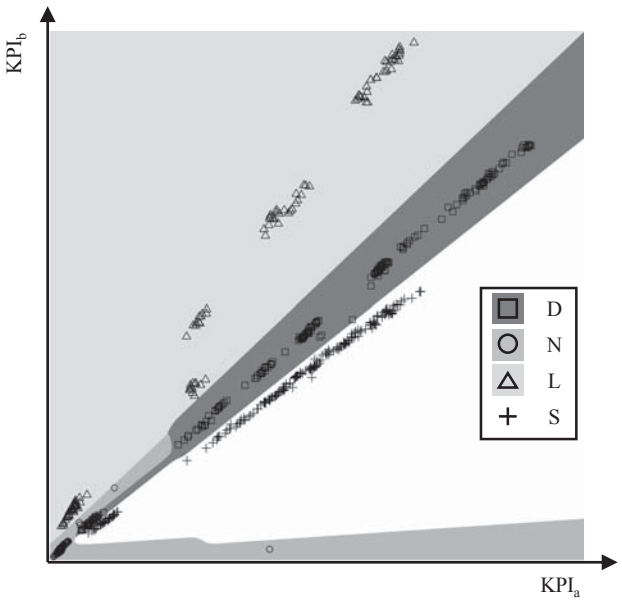

Fig. 5. Classification map of a neural network. The symbols represent the training points of different states, and the corresponding colors represent what state an unlabeled measurement would be classified as in the area.

Training with any of the frequently chosen combinations of KPIs and optimizing the required parameters through 10-fold cross validation the classification accuracy for both the KNN and CART classifiers were above 98%. The results from using the SVM and ANN classifiers will not be presented here such in detail, because we feel that by leaving out the parameter optimization process the chosen KPI sets might not represent the best solutions. Still, as can be seen on Fig. 3, the unoptimized ANN classifier had the best accuracy during the feature selection process. This also carried over to the final classifier, where training with two of the most frequently chosen and using proper optimization the classification accuracy of the neural network was 99.88% on the training set. The classification map of this classifier can be seen on figure 5.

V. Conclusion

In this paper we presented a diagnosis system for mobile networks that uses machine learning techniques to identify anomalous states in the network. The system can select the most important performance indicators so that the classification can be done faster and with higher accuracy. Since the classification rules are generated through a learning algorithm, the system can identify multiple user defined states.

As our research shows, even the simplest classification algorithms can provide great accuracy in identifying complex states in a mobile network. In fact, though the more complex classification algorithms did provide better classification accuracy by some degree, we feel that in this case the gain did not warrant the increased runtime. Our advice for anyone trying out a wrapper feature selector is to experiment with a fast and simple classification algorithm first. It is also worth mentioning that the more complex the classifier, the more research someone needs to figure out how to properly use it.

The presented system is independent of the network, the OSS and the provided KPIs. As long as the classifiers can differentiate the network states with the data provided, the diagnosis system should potentially work on any network. The key componenet in making this algorithm work well as an anomaly detector is supplying the system with expert knowledge about the anomalies. This translates to creating a good training data set for the feature selection algorithm. This can be either generated through measurements on a test network or simulation, or created in real time by labeling measurements from previously identified anomalous network states. After running the feature selection algorithm on the training data, the trained classifiers are ready to use in real time for anomaly detection. The detected states could then be used in network optimization or expansion tasks by the operator, or in self-optimization and self-healing tasks by the network itself.

References

[1] E. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of Machine Learning*, C. Sammut and G. Webb, Eds. Springer US, 2010, pp. 257–258. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-30164-8_192

[2] S. Davies and S. Russell, "Np-completeness of searches for smallest possible feature sets," in *AAAI Symposium on Intelligent Relevance*. AAAI Press, 1994, pp. 37–39.

[3] K. A. De Jong and W. M. Spears, "Using genetic algorithms to solve np-complete problems," in *Proceedings of the Third International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 124–132. [Online]. Available: http://dl.acm.org/citation.cfm?id=93126.93172

[4] G. Ciocarlie, U. Lindqvist, S. Novaczki, and H. Sanneck, "Detecting anomalies in cellular networks using an ensemble method," in *Network and Service Management (CNSM), 2013 9th International Conference on*, Oct 2013, pp. 171–174.

[5] R. Barco, V. Wille, and L. Díez, "System for automated diagnosis in cellular networks based on performance indicators," *European Transactions on Telecommunications*, vol. 16, no. 5, pp. 399–409, 2005. [Online]. Available: http://dx.doi.org/10.1002/ett.1060

[6] G. A. Barreto, J. C. Mota, L. G. Souza, R. A. Frota, L. Aguayo, J. S. Yamamoto, and P. E. Macedo, "A new approach to fault detection and diagnosis in cellular systems using competitive learning." [Online]. Available: http://www.laps.ufpa.br/aldebaro/classes/04mineracao2sem/seminarios/sbrn2004-cellular-fault.pdf

[7] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," 1997. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=671091

[8] R. Kohavi and G. H. John, "Wrappers for feature subset selection," 1997. [Online]. Available: http://ai.stanford.edu/~ronnyk/wrappersPrint.pdf

[9] J. Sepulveda-Sanchis, G. Camps-Valls, E. Soria-Olivas, S. Salcedo-Sanz, C. Bousono-Calzon, G. Sanz-Romero, and J. Marrugat de la Iglesia, "Support vector machines and genetic algorithms for detecting unstable angina," in *Computers in Cardiology, 2002*, Sept 2002, pp. 413–416.

[10] D. R. Eads, D. Hill, S. Davis, S. J. Perkins, J. Ma, R. B. Porter, and J. P. Theiler, "Genetic algorithms and support vector machines for time series classification," pp. 74–85, 2002. [Online]. Available: http://dx.doi.org/10.1117/12.453526

[11] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[12] L. Breiman, *Classification and regression trees*. Belmont, Calif: Wadsworth International Group, 1984.

[13] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1009715923555

[14] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[15] P. Lanzi, "Fast feature selection with genetic algorithms: a filter approach," in *Evolutionary Computation, 1997., IEEE International Conference on*, Apr 1997, pp. 537–540.

[16] T. Blickle and L. Thiele, "A comparison of selection schemes used in genetic algorithms," Gloriastrasse 35, CH-8092 Zurich: Swiss Federal Institute of Technology (ETH) Zurich, Computer Engineering and Communications Networks Lab (TIK, Tech. Rep., 1995.

[17] W. M. Spears and V. Anand, "A study of crossover operators in genetic programming," in *Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems*, ser. ISMIS '91. London, UK, UK: Springer-Verlag, 1991, pp. 409–418. [Online]. Available: http://dl.acm.org/citation.cfm?id=646353.691341