Delft University of Technology
Software Engineering Research Group
Technical Report Series

Improving the Diagnostic Capabilities of a Performance Optimization Approach

Cor-Paul Bezemer, Andy Zaidman

Report TUD-SERG-2013-015

TUD-SERG-2013-015

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Improving the Diagnostic Capabilities of a Performance Optimization Approach

Cor-Paul Bezemer
Delft University of Technology
the Netherlands
c.bezemer@tudelft.nl

Andy Zaidman
Delft University of Technology
the Netherlands
a.e.zaidman@tudelft.nl

***Abstract*—Understanding the performance of a system is difficult because it is affected by every aspect of the design, code and execution environment. Performance maintenance tools can assist in getting a better understanding of the system by monitoring and analyzing performance data. In previous work, we have presented an approach which assists the performance expert in obtaining insight into and subsequently optimizing the performance of a deployed application. This approach is based on the classification results made by a single *classifier*. Following results from literature, we have extended this approach with the possibility of using a set (*ensemble*) of classifiers, in order to improve the classification results. While this ensemble is maintained with the goal of optimizing its accuracy, the completeness (or *coverage*) is neglected.**

In this paper, we present a method for improving both the coverage and accuracy of an ensemble. By doing so, we improve the diagnostic capabilities of our existing approach, i.e., the range of possible causes it is able to identify as the bottleneck of a performance issue. We present several metrics for measuring coverage and comparing two classifiers. We evaluate our approach on real performance data from a large industrial application. From our evaluation we get a strong indication that our approach is capable of improving the diagnostic capabilities of an ensemble, while maintaining at least the same degree of accuracy.

I. Introduction

Understanding the performance of a system is difficult because it is affected by every aspect of the design, code and execution environment [1]. Therefore, tools are being developed which assist experts in understanding the performance of a system. A subset of these tools [2], [3], [4], rely on machine learning techniques to correlate low-level system measurements with high-level service objectives (SLOs). A part of these techniques, the so-called supervised learning techniques [5], use part of the monitored data as input for training the *classifier* used in the performance maintenance system. This classifier can then analyze newly monitored input and reason about, or predict the performance of the system. The quality of the analysis is dependent on the classifier. On the one hand, it is important that the classification made by the classifier is accurate. On the other hand, the range of possible diagnoses the classifier can make, its *coverage*, defines the applicability of the approach on real world problems.

Zhang et al. [4] and Tan et al. [6] showed that the quality of a classification can be improved by using an ensemble of classifiers. In an ensemble, classifiers work together to take a better decision than they would on their own. For example, Zhang et al. select the most suitable classifier for classifying a set of measurements, which works well for analyzing performance anomalies. While Zhang et al. and Tan et al. propose methods for increasing the accuracy of the classifier ensemble, these methods do not take the diagnostic capabilities of the ensemble into account. In this paper, we present an approach which aims at improving both the coverage and accuracy of a classifier ensemble. By improving the coverage of the ensemble, we improve its diagnostic capabilities and we also improve the level of understanding an expert can get from the performance analysis results. In short, we make the following contributions:

1) A set of metrics for defining and calculating the coverage of a classifier or ensemble
2) An ensemble maintenance approach for improving both accuracy and coverage of a classifier ensemble
3) An industrial case study in which we evaluate our approach and show that it is capable of improving the diagnostic capabilities of an ensemble, while maintaining approximately the same degree of accuracy for detecting performance bottlenecks in the system

We do so by extending our approach for performance optimization of deployed applications [7]. Because we focus on finding performance optimizations instead of anomalies, we let classifiers work together in an ensemble by letting them vote and subsequently select the classification which has received the most votes. As a result, the ensemble is more likely to find structural bottlenecks.

In the rest of this paper, we first present our problem statement in Section II. In Section III, we give a brief overview of the approach we extended to come to our contributions. We discuss our approach which aims at improving both coverage and accuracy of a classifier ensemble in Section IV. We present and discuss the evaluation results of our industrial case study in Section V, VI and VII. We present related work in Section VIII and conclude our work in Section IX.

II. Problem Statement

Many performance maintenance approaches are based on supervised learning techniques, i.e., they must be trained using labeled data to be able to reason about the performance of a system. During the training, a (trained) *classifier* is generated which is able to classify new performance data into certain classes. An advantage of using supervised learning instead of

unsupervised learning, is that you have more control over what should be considered abnormal cases in the training data. This is interesting for performance optimization approaches, as it allows you to find structural performance issues, which are possibly missed by unsupervised learning.

A classifier can be trained using any training set specified by a performance expert, but the quality of the classification results can be very different. As the reasoning (or *diagnosing*) part of the approach often relies on this classification, the quality of the diagnosis depends on the quality of the classification as well. The quality of a diagnosis is defined by:

1) The *accuracy*, i.e., whether the diagnosed issue is indeed an issue.
2) The *completeness*, i.e., whether the diagnosis provides a complete description of the cause of the issue, giving the expert enough details to start an investigation.

Existing methods for increasing the accuracy of a diagnosis by using an ensemble of classifiers instead of a single classifier [4], [6] do not consider the completeness of the diagnosis. In this paper, we propose a method which is based on Zhang et al.'s method for maintaining a classifier ensemble. However, our method aims at improving both the completeness and accuracy of the diagnosis. A metric in which the completeness of a diagnosis can be expressed is coverage. The coverage of a classifier or ensemble is the set of features it uses to take a decision. In a performance monitoring system, this set may, for example, consist of a subset of the monitored performance metrics. The coverage of a classifier tells us something about its diagnostic capabilities: the more features a classifier or ensemble covers, the wider the range of possible diagnoses it can make, hence, the more complete the diagnosis can be. We focus on the following research question:

RQ. *How can we improve the coverage of a classifier ensemble, while maintaining at least the same degree of accuracy?*

In previous work [7], we have presented an approach for performance optimization of deployed Software-as-a-Service (SaaS) applications. In this paper, we extend this approach with support for a classifier ensemble maintained using our approach for accuracy and coverage improvement. In the next section, we first give background information about our performance optimization approach.

III. Background

In previous work [7] we presented an approach that allows to identify and analyze possible *performance improvement opportunities* (PIOs) in a software system using performance metrics [1] only. To do this, our approach uses a system-specific classifier, generated from data collected during the training phase. The performance of a system is often characterized by unexpected combinations of performance metric values, rather than following simple rules of thumb [3]. Therefore, we generate association rules [8] from the monitored data[2], as these are capable of representing such complex combinations.

[1]See http://www.st.ewi.tudelft.nl/~corpaul/eollist.txt for the list used in our case study.

[2]Due to the limited amount of space, we refer the reader to [7] for a thorough discussion of the approach.

Table I depicts a rule set and a set of new measurements for a sample (fictituous) system. The sample system consists of server $S1$ with two monitored performance metrics CPU (% CPU Utilization) and MEM (% Memory Usage), and server $S2$ with one monitored performance metric CPU. In [7], we devised an approach which generates such association rules based on the response time. Central to our approach is the `SARatio` metric. The `SARatio` (or *Slow-to-All-actions-ratio*) for a time interval t is defined as:

$$SAratio_t = \frac{|SLOW_t|}{|SLOW_t| + |NORMAL_t|}$$

We define an action[3] as slow (i.e., a member of $SLOW_t$) when it belongs to the 15% slowest actions in terms of response time for a particular user of a particular application (or feature) for a time interval t. We consider the other 85% of the actions as a member of the $NORMAL_t$ class. We calculate the response time per user and per application to make sure that actions that have greater resource demands are not automatically considered slow. An example of this is a bookkeeping system: report generation will take longer for a company with 1000 employees than for a company with 2 employees. When using average response time as threshold setting for this action, the threshold will either be too high for the smaller company or too low for the larger company.

We calculate the `SARatio` for all time intervals of the training period using a sliding window approach, in which the window contains all actions made during time interval t. As we now have a `SARatio`-value for all monitored time intervals, we can identify when the system was running relatively slow. We define the following thresholds for the `SARatio`, such that we can classify system load for each interval:

- $high$: system load is typically too high, which makes it perform slow (top 5% `SARatio` values)
- med: system load may become or may just have been problematic (medium 10% `SARatio` values)
- low: system load is non-problematic (low 85% `SARatio` values)

From this definition, classifications which fall into the $high$ class form the most interesting situations, the PIOs, with respect to performance optimization. Using the `SARatio` class and the performance metric values monitored during a time interval, we use the JRip algorithm from the WEKA toolkit [10] to mine association rules which classify performance measurements into one of the three `SARatio` classes.

Our PIO analysis approach exploits the association rules used during the classification process of the PIO detection to find starting points for exploring possible performance improvement opportunities. The goal of our approach is to analyze the information in the rules matched by a measurement and detect clusters of performance metrics that help decide on which server, hardware or software components we must start looking for performance improvements.

Our approach uses a so-called *rule coverage matrix* m.

[3]An action is the activation of a feature by the user. A feature is a product function as described in a user manual or requirement specification [9].

TABLE I
SAMPLE RULE SET AND PERFORMANCE MEASUREMENTS

Sample association rule set	**Sample measurements**			
1 S1_CPU>80 & S2_CPU>60 → $high$	t	S1_CPU	S1_MEM	S2_CPU
2 S1_CPU>70 & S1_MEM>70 → $high$	0	40	60	50
3 S1_CPU>90 → $high$	1	95	60	50
4 S1_MEM<30 → med	2	98	80	50
5 else → low	3	98	95	80
	4	98	80	50
	5	40	25	50

The rows of this matrix contain the performance metrics, the columns depict measurements. The first column, representing the first measurement is initialized to 0. Each time a new measurement is received, the last column of m is copied and the following algorithm is applied:

- Increase $m_{i,j}$ if performance metric i is covered by a $high$ rule at measurement j.
- Leave $m_{i,j}$ equal to $m_{i,j-1}$ for a med rule
- Decrease $m_{i,j}$ if performance metric i is covered by a low rule at measurement j, with a minimum of 0

We update the value of every $m_{i,j}$ only once for every measurement, even though multiple covering rules may contain the same performance metric. The rationale behind building the rule coverage matrix this way is the following:

1) The rule set describes all known cases of when the system was performing slowly.
2) We expect all measurements made during a PIO to be covered by the same, or similar rules when they are classified. The reason for this is that performance metric values are in general relatively stable, which means that abnormal values of (combinations of) performance metrics will be exhibited for a longer period of time, i.e., throughout the PIO.
3) When entering this into the rule coverage matrix this way, higher values in m will appear because these values will be increased for performance metrics which occur in adjacent measurements.
4) Eventually, clusters of higher values in m for performance metrics on specific hardware will appear.
5) These clusters can be used to do performance reengineering, e.g., pinpointing a bottleneck component.

The following example illustrates this. Table II shows the resulting m after applying our approach to Table I. We can see a cluster of higher values at server $S1$ at $t = 4$ and $t = 5$, indicating this server may be a bottleneck.

TABLE II
RULE COVERAGE MATRIX FOR TABLE I

metric \ t	0	1	2	3	4	5
S1_CPU	0	1	2	3	4	4
S1_MEM	0	0	1	2	3	3
S2_CPU	0	0	0	1	0	0
covered by rules #	5	3	2,3	1,2,3	2,3	4

Our approach for PIO analysis relies on the contents of the rule set used by the classifier. If this rule set contains rules that cover two servers, we can identify one of those two servers as a bottleneck only. Therefore, a higher rule set coverage can lead to a better bottleneck diagnosis.

IV. OUR APPROACH

In this section, we present our approach for improving both accuracy and coverage of a classifier ensemble. First, we will discuss the metrics used to calculate the *accuracy*, and *coverage* of a classifier and ensemble. In addition, we will present metrics for the *contribution* of a classifier. The contribution is a metric for describing how much a classifier would improve the coverage of an ensemble, if it were a member of the ensemble. After this, we will present our approach for accuracy and coverage improvement of a classifier ensemble, which uses these metrics.

A. Accuracy

The accuracy of a measurement system is the degree of closeness of measurements of a quantity to that quantity's actual (true) value[4]. We make a distinction between *accuracy* and *balanced accuracy*. Accuracy is the number of times the classifier or ensemble classified a measurement correctly. For systems in which the occurence of certain classes is rare, this metric can give a false judgement of the system. For example, in our approach, the *low* class represents approximately 85% of all classifications, as we expect the load on a system to be non-problematic in most cases [7]. A classifier which would always classify a measurement as *low*, would have an accuracy of 85%, but it would still be unusable in a production system. Hence, we use the balanced accuracy [11], which is capable of dealing with such imbalances in the data set by taking the true and false positives (*TP* and *FP*) and true and false negatives (*TN* and *FN*) into account. The balanced accuracy BA can be calculated as follows:

$$\text{BA} = \frac{0.5 * TP}{TP + FN} + \frac{0.5 * TN}{TN + FP}$$

The classifier mentioned which has an accuracy of 85%, would have a BA of approximately 0.5.

B. Coverage

In order to improve the coverage of a classifier ensemble, we must be able to compare the coverage of two classifiers, so that we can decide whether adding (or removing) a classifier to the ensemble will increase (or decrease) its coverage. In this section, we introduce metrics for defining the *coverage* of a classifier or ensemble (Section IV-B1) and for the *contribution* of a classifier (Section IV-B2).

Throughout this section, we will use the association rule sets in Table III as an example to illustrate the metrics. These rule sets were synthetically crafted for demonstration purposes. Note that we use the terms association rule set and classifier interchangeably throughout this paper.

TABLE III
SAMPLE RULE SETS

Complete set of monitored performance metrics (M_{ALL}):
S1_CPU, S1_MEM, S2_CPU, S3_CPU
Complete set of monitored servers (S_{ALL}):
S1, S2, S3

Rule set/Classifier A	**Rule set/Classifier B**
1 S1_CPU>80 & S2_CPU>60 → $high$	1 S1_CPU>80 & S2_CPU>60 → $high$
4 S1_MEM<30 → med	1 S3_CPU>90 → $high$
5 else → low	else → low

[4]http://en.wikipedia.org/wiki/Accuracy_and_precision

1) Classifier Coverage Metrics: An important property of a classifier is the set of performance metrics or hardware components it covers. As explained in Section III, the contents of the rule set are used to perform bottleneck analysis on the system. As a result, the larger the set a classifier covers, the more precise the bottleneck diagnosis can be. We propose to exploit this to improve coverage of an ensemble. For example, two classifiers which cover a disjoint set of servers could be complementary.

a) Metrics Coverage Percentage (MCP): The MCP is the percentage of performance metrics of the complete set of monitored metrics M_{ALL} that a rule set R covers. To calculate this percentage, we compose set M_R containing all performance metrics used in rule set R. After this, we calculate which percentage of the complete set M_R covers:

$$\texttt{MCP} = \frac{|M_R|}{|M_{ALL}|} * 100$$

For our example rule sets, this gives:

$$\begin{aligned} M_A = &\quad \{S1\_CPU, S1\_MEM, S2\_CPU\} \\ \texttt{MCP for rule set A} = &\quad \frac{3}{4} * 100 = 75\% \\ M_B = &\quad \{S1\_CPU, S2\_CPU, S3\_CPU\} \\ \texttt{MCP for rule set B} = &\quad \frac{3}{4} * 100 = 75\% \end{aligned}$$

b) Server Coverage Percentage (SCP): The SCP is the percentage of servers of the set of monitored servers S_{ALL} a rule set R covers. SCP is defined similarly to MCP:

$$\texttt{SCP} = \frac{|S_R|}{|S_{ALL}|} * 100$$

For our example rule sets, this gives:

$$\begin{aligned} S_A &= \{S1, S2\} \\ \texttt{SCP for rule set A} &= \frac{2}{3} * 100 = 66,7\% \\ S_B &= \{S1, S2, S3\} \\ \texttt{SCP for rule set B} &= \frac{3}{3} * 100 = 100\% \end{aligned}$$

The MCP and SCP are an indication of the completeness of a rule set. Note that they are not a metric for describing how accurate the rule set can classify new performance measurements. However, they can help with improving the coverage of an ensemble, as the ideal ensemble can cover up to 100% of M_{ALL} and S_{ALL}. Therefore, we define $\texttt{MCP}_{ensemble}$ as the percentage of metrics of the complete set of all monitored metrics that all classifiers in the ensemble cover together. Likewise, we define $\texttt{SCP}_{ensemble}$ as the percentage of servers the ensemble covers. To calculate $\texttt{MCP}_{ensemble}$, we take the union of M_R of every classifier R in the ensemble. We can calculate $\texttt{SCP}_{ensemble}$ by taking the union of S_R of every classifier R in the ensemble. The $\texttt{MCP}_{ensemble}$ and $\texttt{SCP}_{ensemble}$ can be used to decide whether the coverage of the ensemble increases after adding a classifier. Throughout the rest of this paper, we will address the monitored performance metrics and servers as features.

c) Classifier Coverage Vector (CCV): To be able to compare classifiers with each other, we propose to use a vector representation of their contents. The classifier coverage vector (CCV) is such a representation. For a classifier A, we create a vector V_c with $|M_{ALL}|$ elements with value zero, and set value V_{m_i} to $true$ (1) if M_{ALL_i} is in M_A. In addition, we create a vector V_s with $|S_{ALL}|$ elements with value zero, and set value V_{s_i} to 1 if S_{ALL_i} is in S_A. We then concatenate V_c and V_s to create $\texttt{CCV}_A$. For our example set A this gives:

$$\begin{aligned} V_c &= [1;1;1;0] \\ V_s &= [1;1;0] \\ \texttt{CCV}_A &= [V_c; V_s] = [1;1;1;0;1;1;0] \end{aligned}$$

d) Ensemble Coverage Frequency Vector (ECFV): To maintain a classifier ensemble, we must know what the current ensemble coverage is. To keep track of this, we maintain the ECFV, which contains the number of classifiers in the ensemble that cover a feature. We start with an empty ECFV, i.e., all elements are set to zero. Everytime a classifier is added to the ensemble, we add its CCV to the ECFV. Likewise, when we remove a classifier from the ensemble, we subtract its CCV from the ECFV. For our example rule sets, this gives:

$$\begin{aligned} \texttt{ECFV} &= [0;0;0;0;0;0;0] \\ \texttt{CCV}_A &= [1;1;1;0;1;1;0] \\ \texttt{ECFV} + \texttt{CCV}_A &= [1;1;1;0;1;1;0] \\ \texttt{CCV}_B &= [1;0;1;1;1;1;1] \\ \texttt{ECFV} + \texttt{CCV}_A + \texttt{CCV}_B &= [2;1;2;1;2;2;1] \end{aligned}$$

Note that in contrast to the CCV where the elements represent booleans, the elements in ECFV represent the number of classifiers in the ensemble that cover a certain feature. By maintaining the ECFV, we can easily keep an administration of the coverage of the ensemble.

2) Classifier Contribution: When selecting the best out of two candidate classifiers for addition to the ensemble, we want to select the classifier which makes the greatest contribution to the coverage of the ensemble. When one of the classifiers adds more new features to the ensemble than the other, i.e., causes the largest increase in $MCP_{ensemble}$ and $SCP_{ensemble}$, the decision is clear. Note that this number is the difference in the number of zeros before and after adding CCV to ECFV for a new candidate, and after subtracting CCV from ECFV for a removal candidate.

However, when both classifiers cause an equal increase in the coverage of the ensemble, we must find out which classifier provides the most valuable contribution. This is the classifier which covers the features with the lowest frequency in the ECFV, because adding this classifier would allow us more flexibility when removing a classifier later without affecting $MCP_{ensemble}$ and $SCP_{ensemble}$. Imagine the following $\texttt{CCV}_C$ and $\texttt{CCV}_D$:

$$\begin{aligned} \texttt{ECFV} &= [2;1;2;1;2;2;1] \\ \texttt{CCV}_C &= [1;0;0;0;0;0;0] \\ \texttt{CCV}_D &= [0;1;0;0;0;0;0] \end{aligned}$$

Adding classifier C to the ensemble would increase the coverage of $S1\_CPU$ (the first element of ECFV), which is already covered by two classifiers. Adding classifier D to the ensemble would increase the coverage of $S1\_MEM$ (the

second element of ECFV), which is covered by one classifier in the ensemble only. Therefore, classifier D would make the greatest contribution to the ensemble.

We use the cosine similarity [12] for finding the classifier with the greatest contribution. The cosine similarity measures the cosine between two vectors to describe their similarity; it is 1 for vectors that are exactly the same ($\cos 0$) and -1 for vectors that are exactly the opposite ($\cos 180$). We chose to use the cosine similarity over other similarity measures for its easy interpretation. By calculating the cosine similarity of the ECFV and the CCV of the candidate classifiers, we can calculate which CCV is the least similar to the ECFV. Because this vector covers features which are covered less in the ECFV than the other would, its contribution is greater. The (cosine) $similarity$ for two vectors V_1 and V_2 is defined as follows:

$$similarity_{V_1,V_2} = \cos\theta \quad = \quad \frac{V_1 \cdot V_2}{||V_1||\,||V_2||}$$

with $V_1 \cdot V_2$ being the Euclidean dot product formula and $||V_1||$ the norm for vector V_1:

$$V_1 \cdot V_2 = \sum_{i=1}^{n} V_{1_i} V_{2_i} \qquad ||V_1|| = \sqrt{\sum_{i=1}^{n} (V_{1_i})^2}$$

For classifiers C and D, $similarity_{\text{ECFV,CCV}_C}$ and $similarity_{\text{ECFV,CCV}_D}$ are calculated as follows:

$$\begin{array}{ll} \text{ECFV} \cdot \text{CCV}_C = 2 & ||\text{ECFV}|| = \sqrt{19} \\ \text{ECFV} \cdot \text{CCV}_D = 1 & ||\text{CCV}_C|| = \sqrt{1} \\ & ||\text{CCV}_D|| = \sqrt{1} \end{array}$$

$$similarity_{\text{ECFV,CCV}_C} = \tfrac{2}{\sqrt{19}} \qquad similarity_{\text{ECFV,CCV}_D} = \tfrac{1}{\sqrt{19}}$$

This shows that CCV_D is indeed less similar to ECFV than CCV_C, making D the classifier to select for addition as it makes the greatest contribution to ECFV.

Algorithm 1 MAINTAINENSEMBLE(e, d, n)

Require: Ensemble e, DataBuffer d, maximum ensemble size n (0 = unlimited)
Ensure: Trains a new classifier c whenever d is full, and adds it to e when c can increase the coverage of e.

1: **while** !$d.isFull()$ **do**
2: $d.waitForNewData()$
3: **end while**
4: c = $d.trainNewClassifier()$
5: ADDCLASSIFIER(c, e, n)
6: $d.resetBuffer$

Algorithm 2 CALCNEWFEATURES(c, e)

Require: Classifier c, Ensemble e
Ensure: Returns the number of new features c contributes to e

1: **if** $c \in e$ **then**
2: $ecfvOld$ = $e.ecfv - c.ccv$
3: **return** $cntZeros(ecfvOld) - cntZeros(e.ecfv)$
4: **else**
5: $ecfvNew$ = $e.ecfv + c.ccv$
6: **return** $cntZeros(e.ecfv) - cntZeros(ecfvNew)$
7: **end if**

Algorithm 3 ADDCLASSIFIER(c, e, n)

Require: Candidate classifier c, ensemble e, maximum ensemble size n (0 = unlimited)
Ensure: If there is enough space in the ensemble, and c has high enough BA, c is added. If there is not enough space, and the BA of c is at least as high as another classifier in the ensemble, a tiebreak is started to decide which classifier should be in the ensemble.

1: **if** $c.getBA() < e.getMinBA()$ **then**
2: **return**
3: **end if**
4: **if** $e.size() < n$ **or** $n == 0$ **then**
5: $e.addClassifier(c)$
6: **else**
7: **for all** $cl \in e.getMinBAClassifiers()$ **do**
8: $feats[] = (cl, \text{CALCNEWFEATURES}(cl, e))$
9: **end for**
10: $remCandidate = findMinContr(feats)$
11: **if** $c.getBA() > remCandidate.getBA()$ **then**
12: $e.replaceClassifier(remCandidate, c)$
13: **else if** $c.getBA() == remCandidate.getBA()$ **then**
14: **if** CALCNEWFEATURES$(c, e) > remCandidate.contr$ **then**
15: $e.replaceClassifier(remCandidate, c)$
16: **else if** CALCNEWFEATURES$(c, e) == remCandidate.contr$ **then**
17: **if** $cosSim(c, e) < cosSim(remCandidate, e)$ **then**
18: $e.replaceClassifier(remCandidate, c)$
19: **end if**
20: **end if**
21: **end if**
22: **end if**

C. Ensemble Maintenance

Whenever we monitor new performance data, this data may be used to generate a new classifier. Upon generation of such a new classifier, we must decide whether we should add it to the ensemble or not. Algorithm MAINTAINENSEMBLE depicts the steps necessary for maintenance of the ensemble. A data buffer is being kept which contains all the newly monitored data. Whenever the data buffer is full, i.e. it has reached the configured size (for example one day or one week), a new classifier c is trained using the data in the buffer as training set. This classifier is then evaluated on a set of test data and its balanced accuracy BA is calculated. This BA is compared with the BA of the classifiers in the ensemble. If the BA of the new candidate is higher than that of any of the classifiers in the ensemble, it replaces that classifier. If the selection of the classifier to remove or add (i.e., the BA is equal to that of one in the ensemble) is ambiguous, we select the classifier which would give the greatest contribution in coverage if it were in the ensemble. We then select this classifier as the winning candidate and make it a member of the ensemble. Algorithm ADDCLASSIFIER depicts this process.

An important aspect is the amount of space available for the ensemble: this may be either unlimited or limited, e.g., in an embedded system. Note that the ensemble may consist of only one classifier. We will now discuss the maintenance algorithm in the case of unlimited and limited space.

1) Maintenance of an Ensemble with Unlimited Space: In the case of unlimited space (or when there is still space left in the ensemble), the algorithm for ensemble maintenance is depicted by lines 1 to 5 of Algorithm ADDCLASSIFIER. In this case we add the classifier if and only if its BA is at least as high as the BA of one of the classifiers in the ensemble.

2) Maintenance of an Ensemble with Limited Space: Lines 6-22 of ADDCLASSIFIER depict the case where the ensemble is full. We must then select a candidate for removal from the set of classifiers with the lowest BA in the ensemble (*e.getMinBAClassifiers()*). To do this, we must first calculate the number of new features each classifier in the set of removal candidates contributes to the ensemble using CALCNEWFEATURES. The number of new features is calculated as described in Section IV-B2 by counting the number of zeros in the ECFV before and after adding the CCV of the classifier. If the BA of these classifiers is lower than the BA of the classifier c we are trying to add to the ensemble, we can simply replace the classifier with the lowest contribution with c. Otherwise, we calculate the number of new features the candidate classifier c would contribute. If this contribution is greater than that of the removal candidate, we replace that classifier with the new candidate c. If the removal candidate contributes as much new features as c, we must perform a 'tiebreak' to find out which classifier must be in the ensemble. Therefore, we add c only if its cosine similarity with the ECFV is smaller (Sect. IV-B2). This principle is also used to find the removal candidate which contributes the least in line 10 (only we select the classifier with the largest cosine similarity with ECFV as this classifier contributes the least).

V. EXPERIMENTAL SETUP

Subject System We performed a case study on data monitored in Exact Online[5] (EOL), an industrial multi-tenant SaaS application for online bookkeeping with approximately 18,000 users. The application currently runs on several web, application and database servers. It is written in VB.NET and uses Microsoft SQL Server 2008.

Process For a period of 65 days, we collected the data of in total 255 performance metrics[6] on 15 servers, every minute in the same database using a .NET background task. In addition, all requests, including the user who made them and the application that was targeted by the request, were logged. We selected 51 days as training data for the classifier ensemble, and reserved 7 days as new test data to calculate the accuracy of every classifier. The final 7 days were used to calculate the accuracy of the ensemble. The labels for the test set were estimated using the SARatio (see also Section III).

The classifier ensemble maintenance approach was implemented as an extension to the performance optimization approach presented in earlier work [7]. This approach and the extension are implemented in Java and use the WEKA API [10] (in particular the $JRip$ algorithm [8]) to perform the classification. The voting mechanism was implemented as follows. We let each classifier in the ensemble classify the data of every minute that we want to analyze. Each classification made by a classifier for a specific minute counts as a vote. Then, for each minute, we calculate the number of votes for each SARatio class and we select the class with the most votes as the voted classification. The voted classification is the classification made by letting the classifiers in the ensemble work together. In the case of an equal number of votes, we give a preference to the lowest classification. For example, if there are 20 votes, and the med and $high$ class both have received 10 votes, we select med.

Balanced Accuracy In our evaluation we focus on the (mis)classifications of the $high$ class, as this class triggers an action in our approach (Section III). Therefore, we consider measurements wrongly classified as $high$ to be false positives, and measurements which should have been $high$ but were classified differently false negatives. Summarizing, we use the following definitions throughout our evaluation:

- TP: correctly classified as H
- FP: wrongly classified as H
- TN: correctly classified as not H
- FN: wrongly classified as not H

We use these definitions to calculate the balanced accuracy (Section IV-A).

VI. EVALUATION RESULTS

To evaluate our MAINTAINENSEMBLE algorithm, we generated ensembles for each maximum size n ranging from $n = 1$ to $n = 51$. We used 51 classifiers trained on one day of training data to populate the ensemble. We calculated the accuracy and BA for the classifiers by classifying 7 days of test data.

We evaluated our algorithm in 8 situations. Each situation represents a combination of the following parameters:

- *The ensemble maintenance approach*: using only accuracy or using a combination of accuracy and coverage (our approach). In the case of using only accuracy, the oldest classifier was removed from the classifier in case of multiple removal candidates. This parameter is used to investigate whether our algorithm indeed improves the coverage of an ensemble, while keeping the accuracy similar.
- *Precision of classifier accuracy*: 1 or 2 digits after the decimal point. With lower precision, classifiers are more likely to have equal accuracy. As a result, the tiebreak in our algorithm is used more often.
- *The type of accuracy used in our algorithm*: accuracy or BA. As explained in Section IV, BA is more appropriate for our approach. However, as we designed our algorithm to be generic, it should perform with the accuracy metric as well.

In this section, we discuss the coverage and accuracy of the ensembles generated using the described parameters.

A. Coverage Evaluation

In each situation, we calculated $SCP_{ensemble}$ and $MCP_{ensemble}$ at the end of every ensemble generation. The results are plotted in Figure 1 to 4 and summarized in Table IV. The average difference is calculated for those cases in which the values for $SCP_{ensemble}$ and $MCP_{ensemble}$ are not equal.

In Figure 1 and Figure 3, we can see that for 2 digits precision, $SCP_{ensemble}$ and $MCP_{ensemble}$ do not change much. The reason for this is that the high precision results in few conflicts when adding a classifier to the ensemble. Hence, the coverage optimization part of the algorithm is rarely used

[5] http://www.exactonline.nl

[6] See http://www.st.ewi.tudelft.nl/~corpaul/eol_list.txt for a complete list

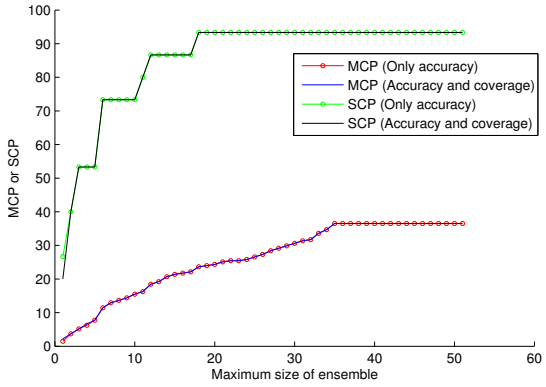

Fig. 1. MCP / SCP (accuracy with 2 digits precision used in algorithm)

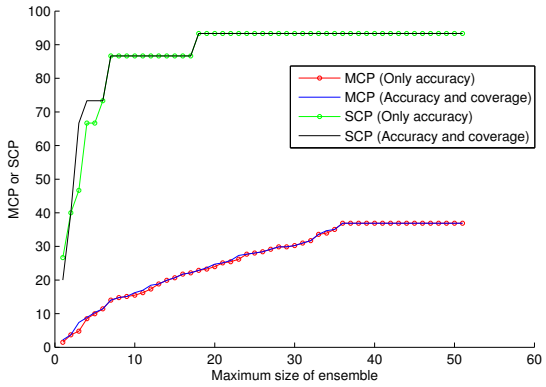

Fig. 2. MCP / SCP (accuracy with 1 digit precision used in algorithm)

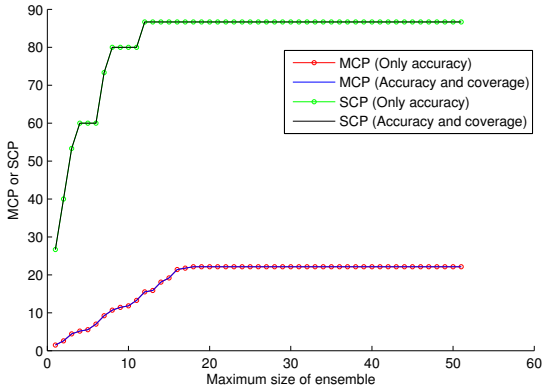

Fig. 3. MCP / SCP (BA with 2 digits precision used in algorithm)

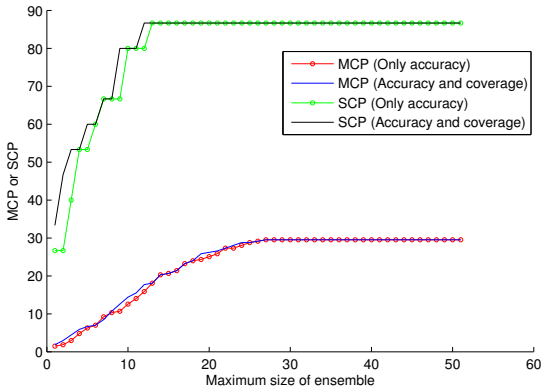

Fig. 4. MCP / SCP (BA with 1 digit precision used in algorithm)

TABLE IV
DIFFERENCES OF MCP AND SCP BETWEEN APPROACHES

Situation		acc. & cov. wins	acc. & cov. loses	equal	avg. diff.
1 digit precision acc.	SCP	3	1	47	6.6625
	MCP	12	0	39	0.8292
2 digits precision acc.	SCP	0	1	50	-6.6700
	MCP	2	0	49	0.5500
1 digit precision BA	SCP	6	0	45	11.1100
	MCP	15	1	35	0.9913
2 digits precision BA	SCP	0	0	51	0.0000
	MCP	0	0	51	0.0000

and the ensembles are generated based on (balanced) accuracy only. In Table IV, the average decrease in $SCP_{ensemble}$ looks larger than the increase of $MCP_{ensemble}$ for the case of 2 digits precision accuracy. However, this is due to the different number of servers and metrics analyzed. As a result, a difference of 6.67% for $SCP_{ensemble}$ means a difference of 1 feature, while a difference of 0.55 for $MCP_{ensemble}$ means approximately 1.4 feature. Because our algorithm tries to optimize the total number of covered features, it may give preference to a classifier which covers more metrics but less servers than the classifiers currently in the ensemble.

For the situations with 1 digit precision, the coverage improvement is done more often. In Figure 2 and 4, we see an improvement of $SCP_{ensemble}$ and $MCP_{ensemble}$ in the ensembles maintained using our approach. For 1 digit precision accuracy, in 4 cases the ensembles generated using our approach covered approximately 1 server more than the ensembles generated using accuracy only. In addition, the ensembles generated using our approach covered an average of approximately 2 metrics more in 12 cases. For 1 digit precision BA, in 6 cases the ensembles generated using our approach covered approximately 1.7 servers more than the ensembles generated using accuracy only. In addition, the ensembles generated using our approach covered an average of approximately 2.5 metrics more in 16 cases. An explanation for the difference in coverage between the ensembles maintained using accuracy and using BA could be that classifiers with high BA have higher coverage than classifiers with high accuracy. In future work, we will investigate whether these metrics are related.

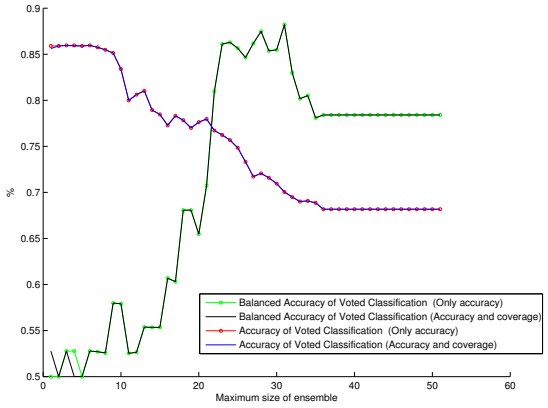

Fig. 5. Accuracy / BA of the voted classification (accuracy with 2 digits precision used in algorithm)

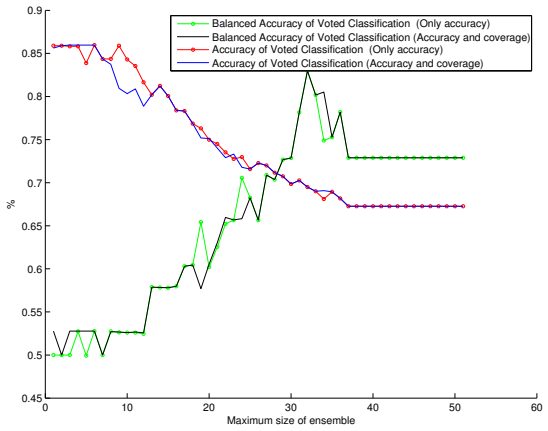

Fig. 6. Accuracy / BA of the voted classification (accuracy with 1 digit precision used in algorithm)

B. Approach Effectiveness Evaluation

To evaluate the effectiveness of our approach, we let each ensemble classify the 7 days of test data using the voting process explained in Section V. We calculate the accuracy and BA of the classification chosen after the voting process was executed by the ensemble. Figure 5 to 8 depict the results of these calculations. The results are summarized in Table V.

The first observation we make is that for the situations with 2 digits precision, there are few differences between the approach which uses accuracy only and our approach. As explained in Section VI-A, this is due to the low number of conflicts when adding classifiers. For the cases with 1 digit precision, Table V, Figure 5 and 7 show that there are more differences. However, the average difference for the ensemble sizes with unequal accuracy and BA is small (at most 1.6%).

Another observation we make, is that the graphs for accuracy and BA are not continuously increasing or decreasing. The reason for this is that we measure the accuracy and BA of the voted classification. Even though we try to increase the average accuracy or BA of the ensemble when adding a classifier, there

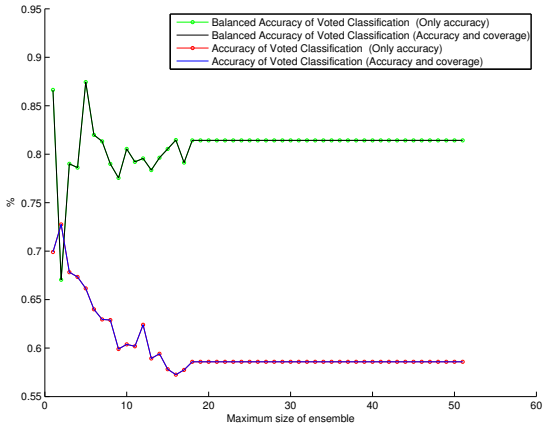

Fig. 7. Accuracy / BA of the voted classification (BA with 2 digits precision used in algorithm)

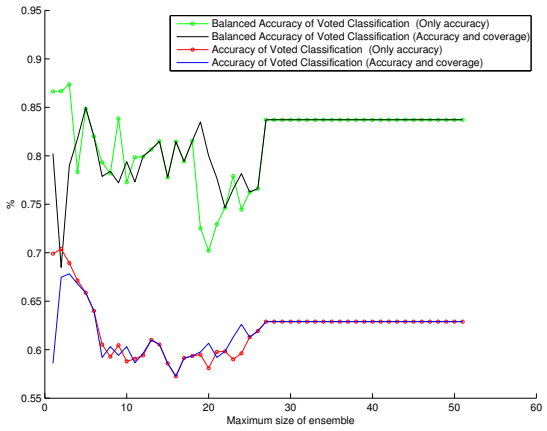

Fig. 8. Accuracy / BA of the voted classification (BA with 1 digit precision used in algorithm)

is no guarantee that a classifier with high accuracy or BA will increase the accuracy or BA of the voted classification. The reason for this is that despite the high accuracy of a classifier, it may accurately classify a different subset of classifications than the classifiers in the ensemble. Hence, it will not always vote for the same classification as the other classifiers in the ensemble during the voting process. As a result, a classifier with high accuracy or BA will not necessarily increase the accuracy or BA of the voted classification, unless the accuracy or BA of the classifier is 100% (which is difficult to achieve). To optimize this, we would have to calculate the accuracy or BA of the voted classification for all possible combinations of classifiers, every time we have trained a new classifier. This would be impractical due to the number of combinations and therefore, we settle for suboptimal results using our approach.

The third observation we make, is the difference between the accuracy and BA of the voted classification. As explained in Section IV-A, this is because we have an imbalanced data set, in which a classifier that classifies all measurements as

TABLE V
DIFFERENCES FOR ACCURACY AND BA OF VOTED CLASSIFICATION BETWEEN APPROACHES

Situation		acc. & cov. wins	acc. & cov. loses	equal	avg. diff.
1 digit precision acc.	Acc.	6	10	35	-0.0091
	BA	12	3	36	0.0023
2 digits precision acc.	Acc.	0	2	49	-0.0017
	BA	1	1	49	0.0000
1 digit precision BA	Acc.	7	8	36	0.0059
	BA	9	7	35	0.0157
2 digits precision BA	Acc.	0	0	51	-0.0054
	BA	0	0	51	-0.0061

low would have high accuracy but low BA. As a result, when maintaining the ensemble using accuracy, classifiers with high accuracy but low BA will be added first and vice versa when using the BA during ensemble maintenance.

C. Conclusion

From our evaluation, we get a strong indication that our approach is capable of improving the coverage of an ensemble, while maintaining a similar degree of accuracy and BA for the voted classification. Table IV shows the coverage clearly increases, compared to the approach which neglects coverage, while Table V shows that the difference of the accuracy and BA on average is very small.

VII. DISCUSSION

A. The Research Question Revisited: 'How can we improve the coverage of a classifier ensemble, while maintaining at least the same degree of accuracy?'

In this paper, we have presented an approach which aims at improving both accuracy and coverage of an ensemble. In an evaluation on an industrial data set, we have shown that ensembles generated using our algorithm, always cover at least the same number of features, compared to an approach which only tries to improve classifier accuracy. In the cases in which $MCP_{ensemble}$ and $SCP_{ensemble}$ improved, the improvement was on average larger than 1 feature.

In addition, we showed in our evaluation that the average accuracy and BA did not change significantly because of our coverage improvement algorithm. Hence, we can conclude, that for this data set, our approach is able of improving the coverage of a classifier ensemble, while maintaining at least the same degree of accuracy.

B. Scalability

Our approach is lightweight and transparent; it requires no modification of application code as measurements are done at the operating system level. The only knowledge our approach needs about the system is the set of features which are being monitored to calculate SCP and MCP.

The data set used was 17 GB in size, containing 163 million records. Running the experiment in which 7 days of data were classified by 51 classifiers and 7 days of data were classifed by the various ensembles took approximately 8 hours. The experiment was ran from code which was not optimized for bulk processing, but is used in production for single classifications. Parallelization offers interesting speed-up opportunities for our approach, as the task performed by the classifiers is completely independent. We did not monitor resource usage statistics during the experiment.

C. Limitations

A limitation of our approach is the fact that we treat features equally when calculating the number of new features contributed by a classifier. While we did not have preference for any type of feature in this experiment, it is possible to assign weights to the types of features.

The fact that we consider all classifiers equal is another limitation of our approach. Our evaluation results could possibly be improved by treating classifiers generated on days during the week and during the weekend differently. Future work must provide more insight into this.

In our work, we used a simple voting algorithm, which selects the classification with the most votes. The same voting algorithm was used throughout the whole experiment. We did not compare other voting algorithms as this is not the core concern of this paper.

D. Different Applications

We assume the classifiers use association rule sets, but any type of classifier (e.g., a decision tree) which explicitly states its decision variables can be used. It is not necessary that classifiers within the ensemble are of the same type.

We have implemented our ensemble maintenance approach in a performance optimization system, but we expect it is applicable for any type of classifier ensemble in which the complete set of monitored features is known.

E. Threats to Validity

We have performed our evaluation on data monitored in an industrial multi-server SaaS application. Due to its outright scale and set-up, this application is likely to be representative of other large-scale SaaS applications, making the monitored data set representative as well. Even so, in future work we will evaluate our approach on other data sets.

Concerning the internal validity of our approach, we acknowledge that we focused on (mis)classifications of the $high$ class during the evaluation. The reason for this is that the $high$ class triggers an action in our approach [7], making false positives and false negatives expensive. Therefore, we focused on these properties by evaluating the BA of the ensembles resulting from our approach.

In our evaluation we did not investigate whether our test set contained any anomalies. While this may influence the accuracy of the classifiers or the ensembles, maintaining a representative test set is difficult in real-world systems due to the manual labour involved [13]. Therefore, we decided to select a week of data and estimate the load classifications using the SARatio. As all classifiers and ensembles were using the same test set, we expect that any anomalies in the set will be filtered out by the voting process.

In addition, we did not evaluate whether the diagnosis itself actually improves due to the improved coverage. This should be done in a user study, similar to the one in [7], making it costly in time. Hence, we will address this in future research.

While we are aware of the existence of different metrics such as support and confidence [14] for comparing rules within rule sets, we did not use these as we were interested in the

coverage of our ensemble. In future work, we will investigate if these metrics can contribute to our approach.

VIII. Related Work

To the best of our knowledge, there is no other work related to performance maintenance in which a classifier ensemble was maintained based on both coverage and accuracy. In this section we discuss some of the research related most to ours.

Zhang et al. [4] present a technique for detecting violations of service level objectives using an ensemble of models. Cohen et al. [15] use this approach to automatically extract a signature from a running system. The ensemble maintenance method used by Zhang et al. and Cohen et al. focuses on classifier accuracy. After training a new classifier, they evaluate it on a synthetic test set, to see if it is more accurate than the classifiers in the ensemble. If the accuracy of the new classifier is higher than all classifiers in the ensemble, it is added to the ensemble. When a classification has to be made, they select the best fitting classifier from the ensemble to make the classification. In this approach, the authors assume unlimited space in the ensemble. As we use a voting process to make a decision, we do not make this assumption, as the number of classifiers in an ensemble influences the voting process. In contrast, using our approach, we are able to find structural bottlenecks, rather than anomalies. In addition, our approach improves the coverage of the ensemble, which is something Zhang et al. and Cohen et al. neglect.

Ryu et al. [13] propose a technique for building classifiers from new data based on changes in the distribution of the monitored data. Their method is based on the assumption that monitored data which belongs to the current distribution of training data can be classified by the ensemble correctly. In future work we will investigate whether this approach is complementary to ours.

ALERT [6] uses an ensemble of classifiers to predict and diagnose performance anomalies. In contrast to ALERT, our system focuses on detecting structural performance improvement opportunities. In addition, ALERT does not improve the coverage of the ensemble.

On the issue of rule set quality improvement, much research has been done in the field of association rule mining [16]. Dudek [14] has defined several metrics for comparing association rule sets. However, these metrics are tailored towards association rule sets with binary features. In addition, the metrics are meant for evaluating the quality of association rule set mining algorithms, instead of comparing the quality of the actual mined rule set. In future work, we will investigate if the metrics presented by Dudek can contribute to our approach.

IX. Conclusion

In this paper we have proposed a technique for improving both coverage and accuracy of a classifier ensemble. The goal of this is to improve the diagnostic capabilities of the ensemble, by broadening the set of possible diagnoses it can make, while keeping at least the same degree of accuracy. Concretely, this means that the set of association rules, which helps an expert understand the performance of a system, gets extended. As a result, the extended rule set can provide a better understanding of the performance as the possibility of giving a more precise diagnosis is likely to increase.

In an evaluation on a large industrial data set we showed that our approach is capable of improving the coverage of an ensemble. Additionally, our approach does this without affecting the ensemble accuracy. We make these contributions:

1) A set of metrics for defining and calculating the coverage of a classifier or ensemble
2) An ensemble maintenance approach for improving both accuracy and coverage of a classifier ensemble
3) An industrial case study in which we evaluate our approach and show that it is capable of improving the diagnostic capabilities of an ensemble, while maintaining approximately the same degree of accuracy for detecting performance bottlenecks in the system

In future work we will investigate whether the improved diagnostic capabilities indeed result in an improved diagnosis, by performing an expert user study.

References

[1] M. Woodside, G. Franks, and D. Petriu, "The future of software performance engineering," in *Future of Softw. Engineering (FOSE)*. IEEE, pp. 171–187.

[2] J. Rao and C.-Z. Xu, "Online measurement of the capacity of multi-tier websites using hardware performance counters," in *Int'l Conf on Distributed Computing Systems (ICDCS)*, 2008, pp. 705–712.

[3] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase, "Correlating instrumentation data to system states: a building block for automated diagnosis and control," in *Proc. of the Symp. on Operating Systems Design & Impl.* USENIX Association, 2004, pp. 231–244.

[4] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of models for automated diagnosis of system performance problems," in *Proc. Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 2005, pp. 644 – 653.

[5] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann, 2005.

[6] Y. Tan, X. Gu, and H. Wang, "Adaptive system anomaly prediction for large-scale hosting infrastructures," in *Proc. Symp. on Principles of Distributed Computing (PODC)*. ACM, 2010, pp. 173–182.

[7] C.-P. Bezemer and A. Zaidman, "Performance optimization of deployed software-as-a-service applications," *Journal of Systems and Software*, to appear. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2013.09.013

[8] W. W. Cohen, "Fast effective rule induction," in *Proc. Int'l Conf. on Machine Learning*. Morgan Kaufmann, 1995, pp. 115–123.

[9] R. Koschke and J. Quante, "On dynamic feature location," in *Proc. Int'l Conf. on Automated Softw. Engineering (ASE)*. ACM, 2005, pp. 86–95.

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[11] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, "The balanced accuracy and its posterior distribution," in *Int'l Conf. Pattern Recognition (ICPR)*. IEEE, 2010, pp. 3121–3124.

[12] P.-N. Tan *et al.*, *Introduction to data mining*. Pearson, 2007.

[13] J. Ryu, M. Kantardzic, and M.-W. Kim, "Efficiently maintaining the performance of an ensemble classifier in streaming data," in *Convergence and Hybrid Information Technology*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7425, pp. 533–540.

[14] D. Dudek, "Measures for comparing association rule sets," in *Artificial Intelligence and Soft Computing*, ser. LNCS. Springer, 2010, vol. 6113, pp. 315–322.

[15] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 105–118, 2005.

[16] S. Shankar and T. Purusothaman, "Utility sentient frequent itemset mining and association rule mining: A literature survey and comparative study," *Int'l Journal of Soft Computing Applications*, vol. 4, pp. 81–95, 2009.

TUD-SERG-2013-015
ISSN 1872-5392