Title

Experience in Building a Comparative Performance Analysis Engine for a Commercial System

Authors

Huang, Peng
Schechter, Craig
Chen, Vincent
et al.

Publication Date

2015-09-28

Peer reviewed

Experience in Building a Comparative Performance Analysis Engine for a Commercial System

Peng Huang, Craig Schechter†, Vincent Chen, Steven Hill,
Dongcai Shen, Yuanyuan Zhou, and Lawrence Saul
University of California, San Diego, † *Teradata Corporation*

Abstract

Performance testing is a standard practice for evolving systems to detect performance issues proactively. It samples various performance metrics that will be compared with a stable baseline to judge whether the measurement data is abnormal. This type of comparative analysis requires domain expertise, which can take experienced performance analysts days to conduct.

In an effort to build an automatic solution for a leading data warehousing company to improve the comparative performance analysis efficiency, we implemented machine learning approaches proposed by existing research. But the initial result has a 86% false negative rate on average, which means the majority of performance defects would be missed.

To investigate causes for this unsatisfying result, we take a step back to revisit the performance data itself and find several important data related issues that are overlooked by existing work. In this paper, we discuss in detail these issues and share our hindsights to address them. With the new learning scheme we devise, we are able to reduce the false negative rate to as low as 16% and achieve a balanced accuracy of 0.91, which enables the analysis engine to be practically adopted.

1 Introduction

1.1 Motivation

Performance is a crucial quality aspect for computer system especially server software. On the other hand, as system evolves, code changes may significantly degrade performance. For example, the Btrfs file system in Linux kernel 2.6.35 introduced a 10 times performance regression that affected a large number of users [1]. To detect performance regression proactively, it is a standard practice to adopt continuous performance testing [2, 3, 10].

Performance testing samples various metrics such as elapsed time and CPU utilization. Practitioners then analyze the produced performance data for a target system by comparing it with a baseline measurement of a stable system. We call this form of analysis **comparative performance analysis**. This analysis task is seemingly easy. For example, if a performance metric deviates from the baseline more than a predefined threshold, e.g., 5%, the performance data is considered to be abnormal, i.e., containing performance regression.

System/Benchmark	# of performance metrics
Teradata Data Warehouse	200
Phoronix Test Suite	165
Chromium	251
Mozilla	278

Table 1: Number of performance metrics sampled in popular benchmark or performance testing for evolving systems.

However, three challenges complicate the task. First, deviation threshold should be chosen carefully. A restrictive threshold can raise false alerts while a relaxed threshold may miss major performance defects. Also, a single threshold is not enough for all metrics because some performance metrics such as network I/O tend to fluctuate more than others like CPU time. Second, comparative analysis needs to factor in variations in performance measurement due to external factors such as caching effect and UNIX environment size [21, 25]. Testing a subject with even the same setting can produce different results. Third, it is often the case that some performance metrics show improvement while others show degradation. A comprehensive performance testing could sample hundreds of metrics as shown in Table 1. Judging which metric is more important and whether the conflicting deviation is an intentional trade-off or serious issue requires expertise and domain knowledge.

As a result, comparative performance analysis is "an art" [16, 17] that is usually conducted by experienced human analysts. This is exactly the practice adopted in

Classifier	False Negative (Rate)
Decision tree	577 (78.0%)
Logistic regression	739 (99.9%)
TAN*	605 (81.8%)
SVM	737 (99.6%)
Random forest	517 (69.9%)

Table 2: Initial false negatives of common classifiers on more than two years' performance data from a commercial system. *: stands for tree-augmented Bayesian network.

a U.S.-based leading data warehousing company, **Teradata**. Manual analysis leverages domain knowledge and experience. But it is also time consuming and sometimes ad hoc. For example, in Teradata, it can take experienced analysts days to a week to analyze the performance result of complex workloads, which may affect timely performance testing and product release. Foo et al. also report similar inefficiency in another enterprise system [14].

To stream the analysis process, recent work proposes classification [11], association rule [14], and statistical techniques [23]. The classification solution implements a *classifier* that learns from past analysis decisions to identify which class (i.e., {normal, abnormal}) a given data instance belongs to. Cohen et al. [11] use the TAN classifier (Tree-Augmented Bayesian Network). The association rule solution mines from labeled data to learn rules about the relationships among performance metrics. If a given data instance violates the rule, it is considered abnormal. The statistical techniques assume certain data distributions and apply statistical tests such as the 3 sigma deviation region to detect anomaly.

While these solutions show great potential, when we apply them to more than two years' performance data from a commercial warehouse test system in Teradata, the initial result is discouraging. Table 2 shows the state-of-the-art classifiers have especially high false negative rate (78–99%). This is problematic because it means, if the automatic solution were used, the majority of performance regression defects would be missed. To investigate causes for the unsatisfying result, we take a step back to revisit the performance data itself. In doing so, we found five important data related issues or properties that were overlooked by us and existing solutions:

- **Invalid data**: Resource contention, hardware malfunctioning or measurement bug can cause performance data to be completely invalid, e.g., negative CPU utilization. Invalid data presents misleading information that harms classification accuracy. In our dataset, including the 9% invalid data in training pollutes the training and, compared with Table 2, incurs 5% additional false negative for classifying the *valid* data . This requires not only cleaning invalid data from the training set, but also adding validation steps before analyzing new performance data.
- **Summary metrics vs. raw metrics**: Performance data contains high level summary metrics like elapsed time and low level metrics such as kernel path time. Performance analysts tend to focus on summary metrics and dive into raw metrics mainly for supporting evidence. Thus it is natural to learn from summary metrics. But our data indicates learning from raw performance metrics can reduce false negative rate to as low as 37%.
- **Collected vs. partitioned data**: In hindsight, we also observe that performance data produced from different test cases in general varies dramatically in characteristics while performance data from test cases of the same workload type tend to share similar properties. Therefore, learning only one classifier from the entire performance data can neutralize the model. With domain knowledge, we partition the data into different groups based on workload types and train a classifier for each group. This simple *grouping* technique proves to be effective that further reduces the false negative rate to 30%.
- **Imbalanced data**: For evolving systems, performance defects are infrequent. Thus, the majority of the performance data is normal. This highly imbalanced distribution causes common classifiers to favor normal data and perform poorly on abnormal data, resulting in low false positive rate but very high false negative rate.
- **Ignorance of baseline**: Comparative performance analysis refers to a prior testing result as baseline. Having a baseline provides an objective performance expectation. However, to simplify the learning task, the baseline data is usually ignored by current solutions. Including the baseline in the model has potential to make analysis more accurate.

§4 details these data issues, hindsights, how they affect the analysis accuracy and efficiency, and how we address them. Due to space limitation, in the following sections we focus on the classification method and representative classifiers (decision tree, logistic regression and TAN). Other classifiers in general comply with the discussion.

With the data issues resolved, evaluation shows our final analysis engine can reduce false negative rate from 86% on average down to as low as 16% and achieve a maximum 0.91 balanced accuracy, a 0.3 improvement. The training time is as low as 3 minutes.

1.2 Threats to Validity

Even though we evaluate our discussion and solution on a commercial warehouse test system that is representative of complex database and storage systems, we

Workload	Ver.	Elapsed Time(sec)	Txn Count	CPU Util.(%)	Kernel Path CPU(ms)	...	Network Msg/Sec	# Disk IO/Txn	
W_1	3.0.0	1230	3000	90.8	12944.54	...	41.35	41689.12	A
W_1	3.0.0	1255	3000	91.6	10899.13	...	45.36	41048.55	
~~W_1~~	~~3.0.0~~	~~1253~~	~~2800~~	~~89.7~~	~~10266.95~~	...	~~44.40~~	~~41987.93~~	
W_1	3.0.0	1261	3000	93.5	10964.34	...	38.24	42100.05	
W_1	3.0.0	1247	3000	92.6	10865.91	...	46.00	42075.47	
W_1	3.0.1	1142	3000	93.7	13700.00	...	48.62	40180.98	B
W_1	3.0.1	1138	3000	91.8	11361.73	...	46.78	39388.56	
W_1	3.0.1	1132	3000	92.4	10697.64	...	49.63	39436.92	
W_2	3.0.0	520	400	95.6	40.80	...	163.68	912.76	C
W_2	3.0.0	515	400	95.8	40.59	...	161.50	961.65	
W_2	3.0.0	513	400	93.5	40.24	...	166.12	945.10	
W_2	3.0.1	607	400	92.7	53.24	...	183.03	915.53	D
W_2	3.0.1	609	400	92.4	61.19	...	171.45	927.76	
W_2	3.0.1	607	400	92.5	61.19	...	181.57	931.89	

Table 3: A contrived example based on real data from Teradata of comparative performance analysis for target version 3.0.1 and baseline version 3.0.0. Data in each group is of the same measurement configuration. The strikeout row is marked as invalid due to its inconsistent transaction count (2800), which is a domain specific rule.

are also aware that theses data issues and hindsights are gained from this specific subject.

We would like to validate them on performance data from other systems. But we are limited by the unavailability of public comprehensive performance data. For example, we attempted to obtain the performance data of Chromium and use its issue tracking system to approximate the ground truth, but we found a lot of performance issue reports refer to internal performance test data that is unavailable to public.

Therefore, we do not claim our discussion applies uniformly to other performance data. However, we believe these issues are important factors to consider when designing similar analysis engines, and sharing our experiences provides a pragmatic exploration of the design spaces.

2 Problem Statement and Goals

In this section, we explain and formulate comparative performance analysis problem in detail.

2.1 Input

A performance measurement of a target system samples an n-dimension performance vector $\vec{m} \in \Re^n$, with each dimension representing a performance metric such as elapsed time and CPU utilization. Attached with $\vec{m}$ is a metadata section $\mathbf{i}$ of k fields, which records the settings about the measurement such as system version, workload type and hardware. In Table 3, the first two columns in each row belong to the metadata section, while the remaining columns form the performance vector $\vec{m}$.

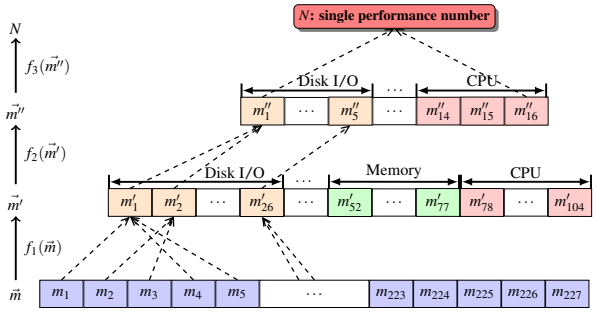

Figure 1: Performance metrics roll-up process: fold detailed performance metrics (in the lower level, $\vec{m}$) into summary metrics (in the higher level, $\vec{m'}, \vec{m''}$) and all the way to a single performance number

For high-dimensional performance vector $\vec{m}$ (e.g., $n \geq 200$), to make manual analysis manageable, $\vec{m}$ may be rolled up by some function $f(\vec{m})$ into a shorter vector $\vec{m'}$ or even a single performance number N [27]. For example, a MB per second high-level I/O metric is summarized from several I/O break-downs and the total I/O time metrics. We refer to $\vec{m}$ as the *raw vector* and $\vec{m'}$ as the *summary vector*. Figure 1 shows the roll-up process for the Teradata test system. In this way, performance analysts can interact with summary vectors most of the time while diving into raw vectors for supporting evidence.

Since a single measurement run could suffer from noises, the measurement is often repeated multiple times to minimize experimental errors. This means the measurement produces a matrix of performance data.

Comparative performance analysis takes as input a pair of performance matrices, $< \mathbf{M}, \mathbf{N} >$, where $\mathbf{M}$ is the target and $\mathbf{N}$ is the baseline. The row count of a perfor-

mance matrix is the number of measurement runs. Each row in the performance matrix comprises of performance vector $\vec{m}$, raw or summary, and metadata section **i**. **i**'s within the each matrix are the same, while **i**'s between baseline and target matrices have all but one field different, referred as comparison field.

A typical comparison field is software version. In this case, the comparative analysis is performance regression analysis. For example, in Table 3, using performance matrix **A** as baseline and **B** as target, the analysis compares the performance of version 3.0.1 for workload W_1 with that of version 3.0.0. If, instead, **C** is used as target, the comparison field is the workload, i.e., comparison between workload W_1 and W_2 for version 3.0.0.

2.2 Output

The output of comparative performance analysis is a boolean value indicating whether target performance matrix **M** is abnormal or not. If the value is true, it indicates **M** *significantly* differs from baseline matrix **N**.

2.3 Analysis

Comparative performance analysis acts as the "black box" that takes two performance matrices as input and outputs a boolean flag. In this sense, the analysis is a two-class classification task. Formally, the task is a mapping function $F: (\langle \mathbf{M}, \mathbf{N} \rangle) \mapsto B$, where **M** is the target performance matrix, **N** is the baseline performance matrix, and B is the binary class variable whose value belongs to $\{0,1\}$ or $\{normal, abnormal\}$.

For systems adopting continuous performance testing, analysts already labeled past performance data. In other words, the task becomes a supervised learning problem [7]. Given T labeled examples $\{(\langle \mathbf{m_1}, \mathbf{n_1} \rangle, b_1), (\langle \mathbf{m_2}, \mathbf{n_2} \rangle, b_2), \ldots, (\langle \mathbf{m_T}, \mathbf{n_T} \rangle, b_T)\}$, the task is to learn the mapping function F.

2.4 Goal

Our primary goal is to design an automated comparative performance analysis engine to conduct the analysis accurately and efficiently. We set the following design targets for the analysis engine in decreasing priorities:

1. **Accurate**: Achieve accuracy competitive with experienced analysts. Otherwise, performance defects would be missed.
2. **Efficient**: Run efficiently within testing cycles. Otherwise, manual analysis would be preferred.
3. **Evolving**: Adapt as system under measurement changes. Otherwise, the solution would not be adopted in continuous performance testing.
4. **Objective**: Minimize human intervention and arbitrary settings. Otherwise, it would be ad-hoc.

3 Dataset and Baseline Result

The dataset we use is from recent two years' performance testing data of a *commercial* system in a leading data warehousing company, Teradata. The performance testing in Teradata uses hundreds of test cases representing various workloads and is conducted regularly. Each test samples a comprehensive set of performance metrics. Testing data will be manually analyzed by performance analysts who judge whether the data is of passing quality by comparing it with the baseline data.

More specifically, the dataset contains 22,964 performance vectors, of which 20,920 are valid (flagged by performance analysts). Each valid performance vector is labeled with either `normal` or `abnormal`. Among the valid vectors, there are 20,180 (96.5%) normal instances and 740 (3.5%) abnormal instances, a 28:1 ratio.

Each performance vector has two forms: the raw data samples 200 low-level performance metrics and the summary data contains 16 high-level performance metrics. Performance analysts mainly look at the summary data and referring to raw data for more evidence.

3.1 Classifiers

With labels available, the core part of a comparative performance analysis engine is a supervised two-class classification task. In our application domain, the features are the performance metrics and the output classes are binary $\{0,1\}$ (or $\{\texttt{normal}, \texttt{abnormal}\}$) with positive class denoting significant difference from baseline.

As supervised learning is a well explored area, we implement classifiers used in state-of-the-art solution [11, 14] as well as other common classifiers such as decision tree and logistic regression. We omit the descriptions for these classifiers and refer readers to [7, 29] for detailed background.

3.2 Measures

Various measures can be used to evaluate a classifier. True positives (TP) are the positive instances classified as positive; True negatives (TN) are the negative instances classified as negative; False positives (FP) are the negative instances classified as positive; False negatives (FN) are positive instances classified as negative.

We are mainly concerned with the *accuracy* measure of a classifier, which is the fraction of instances that are correctly classified, defined as

$$\frac{TP+TN}{TP+FN+TN+FP} \tag{1}$$

However, this overall accuracy measure can be misleading when the data is imbalanced. For example, when there are 99 negative instances and 1 positive instance, a trivial classifier could simply mark every instance as negative and achieve 0.99 overall accuracy. But if identifying the positive instance is important, the 0.99 overall accuracy hides the fact that positive class has 0 accuracy. For evolving systems, performance defects are infrequent. Therefore the majority of data belongs to the negative class. And a false negative has high penalty because it means a performance defect would be unspotted even if testing already reveals it.

To accommodate this issue, the main measure we adopt is **balanced accuracy** [8, 11] that averages the accuracy of both classes. It is formally defined as

$$\frac{1}{2} \cdot (\frac{TP}{TP+FN} + \frac{TN}{TN+FP}) \quad (2)$$

The previous example has a balanced accuracy of $\frac{1}{2} \cdot (\frac{0}{0+1} + \frac{99}{99+0}) = 0.5$.

4 Data Issues and Hindsights

The core part of an automated comparative performance analysis with learning abilities lies in the classifiers used. Adopting state-of-the-art classifiers is merely an engineering effort. However, the initial result as shown in Table 2 is unsatisfying.

Since the classifiers are mature, we devote our attention to the data. As a result, we identify five important issues or properties that are overlooked by us and existing solutions. If not addressed, these issues can significantly impair the analysis accuracy. In this section, we discuss these issues, our hindsights and resolutions. §6 quantifies the accuracy gains for addressing these issues.

4.1 Data Validity

Like any other measurement, performance measurement inevitably contains errors, systematic or random. When there is a serious measurement error, e.g., due to bugs in sampling code or interference from other processes, the data collected can be completely invalid. Examples of invalid performance data include negative elapsed time and inconsistent transaction count (domain specific).

These invalid data instances differ greatly with valid ones. Including them in the training set undermines the classification power. In our dataset, there are 2,044 (9%) invalid performance vectors. Originally, the normal/abnormal labels are only assigned to the metadata, e.g., version A with workload B. In expanding the labels to performance vectors matching the metadata, we did not filter the 9% invalid performance vectors. Consequently, the invalid data pollutes the training process and, compared with Table 2, incurs as high as **5%** additional false negative rate for classifying the *valid* data.

This mistake is due to our carelessness and can be simply fixed. However, it triggers us to also add an explicit validation component in our analysis engine to avoid wasting analysis time on invalid data and polluting *new* learning process. When a new performance vector is fed into the analysis engine, it will first be validated. If it is invalid, further analysis will be skipped.

The validation component can be implemented with some predefined rules. In our case, since performance analysts in the past labeled performance data with valid/ invalid flag, we leverage the labels and implement the validation component with a decision tree classifier.

4.2 Summary Metrics vs. Raw Metrics

At the beginning of developing the analysis engine, we decided to focus on the 16-dimension summary data because human analysts mainly deal with summary data while only drilling down to 200-dimension raw data for more evidences. Since our automatic engine learns from analysts' labeling, it is natural to build models from summary data. The low dimensionality of summary data also makes it easier to manually validate intermediate output from classifiers such as a learned decision tree.

Due to the unsatisfying initial accuracy, we have to revisit this choice of using summary data. Retrying with raw data gives significant accuracy gains (Table 6). We speculate that this is because the summary data is highly condensed from the 200-dimension raw data. Some useful information might be lost during the roll-up process. For example, an I/O metric in the summary data is affected by more than 20 low-level metrics in the raw data. Without these supporting metrics, the classification model using the single I/O metric is likely incomplete.

However, using raw data decreases interpretability and efficiency. For example, the produced decision tree might be hard for analysts to interpret and validate. Given accuracy is our priority (§2.4), we sacrifice interpretability.

But the inefficiency incurred by the high-dimensional raw data must be mitigated because making comparative performance analysis efficient is our original motivation of developing an automatic solution. With the 16-dimension summary data, a decision tree model needs about 8 seconds to be built, whereas the time increases to more than 30 minutes on the 200-dimension raw data.

Therefore, we consider reducing dimensionality of raw data to make training faster by using a common method, Principal Component Analysis (PCA) [17]. To this end, choosing the dimensionality faces trade-off between accuracy and efficiency. Low dimensionality has low training time but may jeopardize accuracy. Higher

dimensionality might improve accuracy but has efficiency penalty. We let practitioners specify constraints such as dimensionality within 100 and the engine automatically searches for the dimensionality that gives best accuracy based on existing data.

4.3 Collected vs. Partitioned Data

Even with raw data, the accuracy still falls short of our expectations. In hindsight, we observe that our dataset, which is collected from more than 300 test cases covering a variety of workloads, has very diverse patterns. But performance data from similar workloads (e.g., write-intensive) often shares similar characteristics.

This leads us to conjecture that part of the low accuracy comes from learning only one model from the entire data that the learned patterns are neutralized. Instead, it might be better to partition the data into groups and learn a per-group classifier. A new performance vector will be predicted with the classifier specific for the group that the vector belongs to. Ideally, each group should correspond to a test case. But this partitioning worsens class imbalance problem (§4.4): more than half of the test cases in our data have fewer than three minority instances.

Instead, we use workload type such as OLTP (Online Transaction Processing) and DSS (Decision Support System) to group the data. This information is embedded in the test case name and documented internally. In this way, the data is partitioned into 20 groups.

While this grouping by workload type strategy is simple and domain specific (requiring knowledge of workload type), to our surprise, it achieves considerable accuracy improvement for all classifiers. Another appealing property of grouping strategy is that it is compatible with other data processing techniques. For example, evaluation shows grouping also improves the accuracy result when combined with undersampling.

The disadvantage of partitioning data into groups and learning separately is that it may weaken common patterns among groups. For example, 100% CPU utilization is problematic for most workloads. To cope with this factor, a transformation matrix from the collected data is constructed first to capture global pattern. Then each group data is transformed using this matrix. The transformed group data will be used to train a group specific classifier. Since the PCA for dimensionality reduction purpose produces projection matrix, we apply it on the collected data to obtain the global transformation matrix.

4.4 Class Imbalance Issue

Our initial classification result shows low false positive rates but very high false negative rates. This corresponds to the skewed class distribution in data, known as class imbalance issue [24]. Imbalanced class distribution impairs standard classifiers because many classifiers are premised on maximizing the overall accuracy and that the classes are of equal importance [24], which causes them to favor the majority instances.

Favoring majority is problematic when detecting the minority is crucial. In performance analysis domain, abnormal performance data is the minority. And missing the anomaly has high penalty: performance defects will remain unnoticed. In our dataset, 96.5% of the data instances are normal. This leads common classifiers to incur high false negative rates.

In general, two approaches are proposed to address the class imbalance issue: the algorithmic approach modifies learning algorithms to be sensitive to the different misclassification costs for different classes [13, 20]; the resampling approach creates artificial minority instances (oversampling) or removes instances of the majority class (undersampling) to balance the distribution.

We experimented with these two approaches and find they can achieve sizeable accuracy improvement, provided the parameters such as cost ratio are set appropriately. This tuning effort is non-trivial. An improper parameter can decrease accuracy significantly as evaluation shows. We allow automatic tuning by further partitioning the training set into two subsets (with 2:1 ratio) : one subset is used to train the model and another is used to try different parameters and calculate accuracies. The optimal parameter will be chosen for using on new data.

4.5 Ignorance of Baseline

Comparative performance analysis uses results from similar measurements as baseline. Having a baseline provides a performance expectation for performance analysts to judge a given result. Formally, the comparative performance analysis task is to learn a mapping function F from training data $\{(\langle \mathbf{m_1}, \mathbf{n_1} \rangle, b_1), \ldots, (\langle \mathbf{m_T}, \mathbf{n_T} \rangle, b_T)\}$ to outputs $\{\hat{y_1}, \ldots, \hat{y_T}\}$.

In our early development, similar to state-of-the-art solutions [8, 11, 12, 14], we simplify the data input to be single performance vector instead of vector pairs. It makes adopting most classifiers easier because pair-wise input is uncommon to a normal classifier. To be specific, the training data is flattened to be $\{(\mathbf{m_1}, b_1), (\mathbf{n_1}, normal), \ldots, (\mathbf{m_T}, b_T), (\mathbf{n_T}, normal)\}$.

We review this simplification to see whether pairing baseline with target can improve analysis accuracy. A straightforward way to is to concatenate the baseline and target data as one vector. Another way is to calculate the deviation percentage of target from baseline as a new vector. The concatenation approach preserves the baseline data but doubles the data dimensionality, which significantly increases training time. The subtrac-

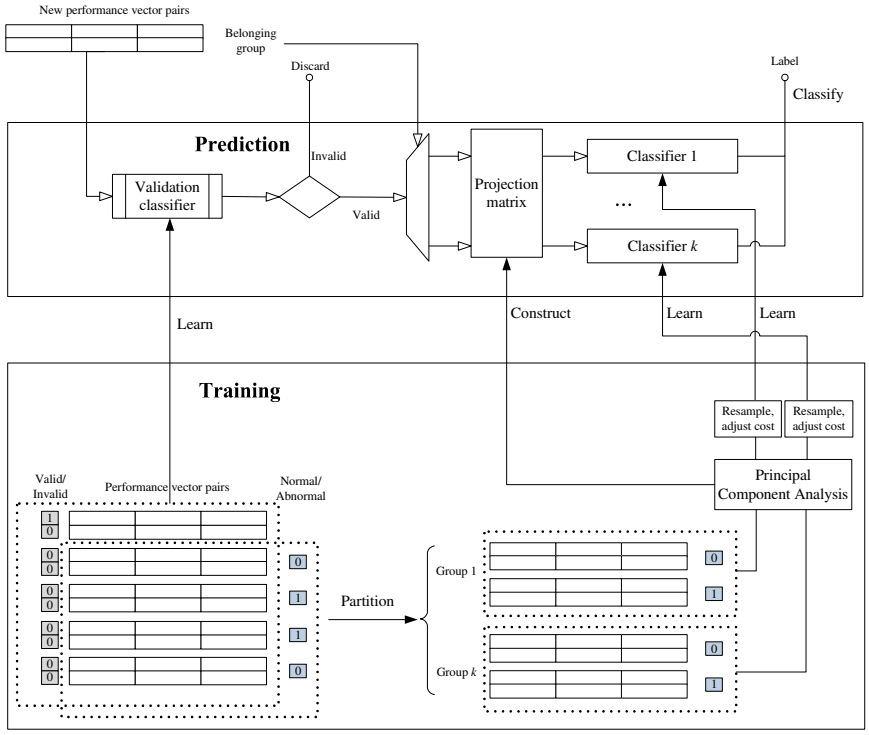

Figure 3: **Training and prediction workflow of the analysis engine.**

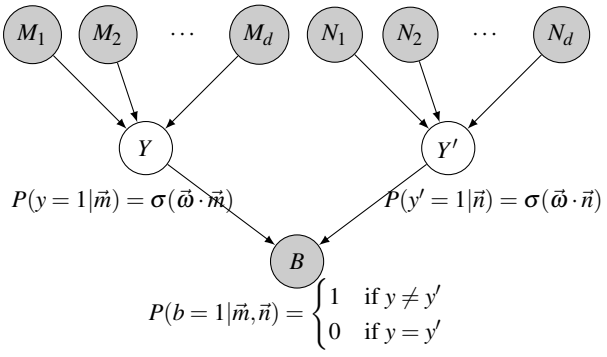

Figure 2: Similarity logistic classifier. Shaded nodes denote observed random variables. Edges denote conditional dependence. B is the class variable with 1 denoting baseline and target vectors are dissimilar.

tion method keeps dimensionality but only looks at deviation percentage , and as evaluation shows, has lower accuracy. This efficiency-accuracy trade-off motivates us to develop a different scheme for handling the baseline.

In particular, we develop a simple *similarity classifier* to learn from examples a mapping function that determines whether a given pair of data is similar or not. Figure 2 shows the model for similarity learning with logistic regression. In a normal logistic regression classifier, $P(y=1|\vec{m}) = \sigma(\vec{\omega}\cdot\vec{m})$ is used to model the conditional probability of y belongs to class 1 given vector $\vec{m}$. Given examples of y and $\vec{m}$, the learning process finds the weight parameters $\vec{\omega}$ that maximizes the likelihood of training examples.

In a similarity classifier, there are two logistic regression sub-models, one for baseline and one for target, which share the same weight parameters $\vec{\omega}$. The Y and Y' are hidden variables whose values are not given. The normal/abnormal labels in the examples are modeled using an additional random variable B that is conditional dependent on Y and Y'. In the simple model, B is 0 (i.e., normal) when Y and Y' have the same values (i.e., similar). Since there are hidden variables Y and Y', we use Expectation Maximization (EM) algorithm [7] to find the best weight parameters $\vec{\omega}$.

5 Comparative Performance Analysis Engine

Having revisited the data issues, we briefly explain the resulting workflow of our comparative performance analysis engine, which is depicted in Figure 3. It incorporates our hindsights and the new learning scheme we devise. In high level, like other machine learning solutions, the engine has a training phase with labeled data and produces classifiers for prediction.

5.1 Training

Our early mistake shows invalid data hurts classification accuracy for even valid data, so it should be excluded from training set. It is also necessary to have a component to validate *new* performance data, since analyzing invalid performance data wastes practitioners' time and pollutes new training process. We implement this component with a decision tree classifier that leverages the validity labels from performance analysts. After the validation classifier is built, the invalid data will be pruned from further training tasks.

Instead of directly applying learning algorithms on valid data, the data is first partitioned into different groups as hindsight suggests the collected data might neutralize anomaly patterns. Our criteria for grouping is the workload type of the test case (embedded in its name) that produces a performance vector. Automatic grouping without domain knowledge is left as future work. To cope with potential loss of common patterns across groups due to partitioning, we construct a single projection matrix using PCA on the collected data and then project the data in each group. Afterwards, the projected per-group data will go through resampling and/or cost adjustment for addressing the class imbalance issue. Finally, we learn a specific classifier for each group.

5.2 Prediction

When a new pair of baseline and target performance vector is collected, the validation classifier will predict whether the new target data is valid or not. If it is valid, depending on which group the performance vector belongs to, in our case its workload type, the data will be transformed with the single projection matrix constructed from training. Then a group-specific classifier will output the label it predicts for the input.

6 Evaluation

We use a standard 10-fold cross-validation for evaluating the classifiers. The main measure used is balanced accuracy ([8, 11], §3.2) to account for the class imbalance situation. Experiments are carried out in a machine with a 2.8GHz quad-core processor and 4GB RAM.

6.1 Baseline Result

Table 4 shows the initial result of evaluating common classifiers on our dataset without addressing the data issues (except that invalid data is already removed) as discussed in §4. The main measure, balanced accuracy, is low (around 60%) for all classifiers due to the very high false negative rate. This is unacceptable to practitioners since it means the majority of performance defects that were manifested in performance testing would remain unspotted because of the inaccurate analysis. Therefore, we focus on improving balanced accuracy by reducing false negatives.

Classifier	Balanced accuracy	False positive	False negative
DTree	0.61	58	577
Logistic	0.50	2	739
TAN	0.59	172	605

Table 4: Initial balanced accuracy, false positive and false negative for running common classifiers (decision tree, logistic regression and tree-augmented Bayesian network) on summary data.

6.2 Overall Improvement

Classifier	Balanced accuracy	False positive	False negative
DTree	0.91 (+0.30)	280 (+222)	122 (-455)
Logistic	0.81 (+0.31)	808 (+806)	253 (-486)
TAN	0.87 (+0.28)	846 (+674)	160 (-445)
*Sim-Logistic**	0.82 (n/a)	454 (n/a)	117 (n/a)

Table 5: Result after resolving all data issues and applying new learning scheme. Changes in brackets are with respect to the baseline result in Table 4. *: the similarity classifier trains on pairs of vectors and therefore is not comparable with other results.

We first show in Table 5 the result by addressing all the data issues and using the proposed new learning scheme. Overall, we achieve a maximum of 0.31 balanced accuracy improvement and reduce the false negative rate to as low as 16%, making the analysis engine practical to be adopted in performance regression testing cycles.

The false positive rates increase by 1–4%, which is acceptable in our scenario. Because compared with manual analysis on the 20,920 data instances, the automatic solution only requires at most 1,426 (true positive + false positive) data instances to be manually validated.

In general we find non-linear classifiers such as decision tree and random forest (not shown) achieve much better balanced accuracy than linear classifier like logistic regression. This suggests a linear model might not fully capture the complex relationship between the labels and performance metrics.

Also note that the result of similarity logistic classifier we develop to factor in baseline vectors is unfortunately not comparable with other classifiers in Table 5. This is because, the original dataset consists of 20,180 performance vectors (target and baseline). Pairing baseline

with target in training theoretically reduces the dataset to 10,090 pairs of vectors. But 1,100 target vectors in the original dataset refer to old baseline data outside the two-year dataset window. This results in a different, pair-wise dataset of 9,540 vector pairs, among which 377 (4.0%) pairs are abnormal. Also, this pair-wise input is difficult to be supplied to common classifiers except by concatenation or subtraction. In §6.7, we compare the similarity classifier with the concatenation or subtraction approach.

In the following sections, we evaluate the effectiveness for each individual resolution and technique.

6.3 Using Raw Metrics

Classifier	Balanced accuracy	False positive	False negative
DTree	0.81 (+0.2)	179 (+121)	282 (-295)
Logistic	0.54 (+0.04)	43 (+41)	674 (-65)
TAN	0.78 (+0.16)	1149 (+388)	277 (-256)

Table 6: Result of using the 200-metric raw data. Changes in brackets are with respect to result in Table 4 obtained using summary data.

Although high-level performance data makes manual analysis tractable and is easier to enforce Service Level Objectives (SLO), we find training with the 200-dimension raw data features gives clearly better accuracy. Table 6 shows the accuracy for using the raw data. For example, decision tree classifier achieves 0.2 accuracy improvement with raw data.

6.4 Grouping

Classifier	Balanced accuracy	False positive	False negative
DTree	0.85 (+0.04)	157 (-22)	223 (-59)
Logistic	0.77 (+0.23)	332 (+289)	322 (-352)
TAN	0.83 (+0.05)	390 (-759)	235 (-42)

Table 7: Result of using grouping technique on raw data. Changes in brackets are with respect to result in Table 6 obtained without grouping.

One of our observations from revisiting the data is that performance data from different test cases but similar workloads often exhibits similar characteristics. This leads us to partition the dataset into 20 groups and learn a per-group classifier. The grouping criteria we use is the workload type embedded in the test case name.

Compared with Table 6, grouping boosts accuracy for all three classifiers. The improvement is most significant (0.23) for logistic regression classifier. We suspect

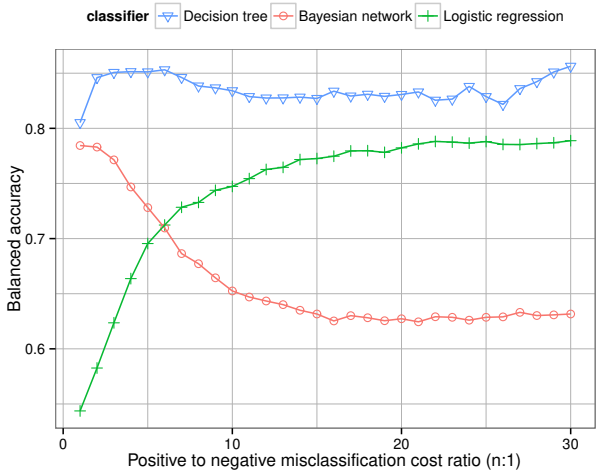

Figure 4: **Balanced accuracy for different cost ratio. We keep the negative class misclassification cost as 1 and increase the cost for positive class. 1:1 is without cost adjustment whose balanced accuracy is in Table 6.**

this is because the logistic classifier is linear. By having different logistic models for different groups, the overall model is non-linear. In comparison, decision tree's overall model is already non-linear even without grouping and therefore benefits less from grouping.

6.5 Cost Adjustment and Resampling

Since our performance dataset is highly skewed in the class distribution, we explore cost adjustment and resampling techniques proposed by existing literature to mitigate the issue. The effectiveness of these two techniques depends on the tuning of the cost ratio parameter and resampling percentage parameter, respectively.

Figure 4 shows the balanced accuracy for different cost ratio adjustment. We can see that adjusting cost help if the ratio is set properly. But the improvement vanishes quickly for decision tree. It even hurts accuracy for Bayesian network classifier. This means non-trivial tuning efforts are needed. We use the method described in §4.4 to automatically set the ratio.

Another technique proposed to address the class imbalance issue is resampling training data to correct the skewness in distribution. Undersampling removes data instances of majority class. Oversampling creates artificial data instances for minority class.

Figure 5 shows the balanced accuracy for different undersampling percentage setting. Leftmost is the original majority to minority ratio (28:1) and rightmost is when majority to minority ratio becomes 1:1. As more instances of the majority class are removed, the balanced

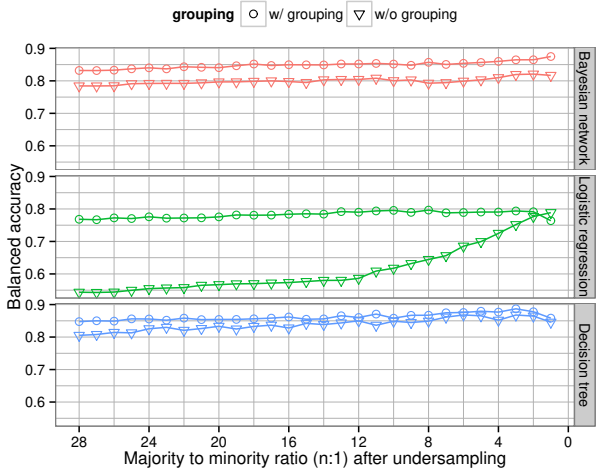

Figure 5: **Balanced accuracy for undersampling. Leftmost is the original majority to minority ratio (28:1) without undersampling. The "w/ grouping" series are for combining undersampling with grouping.**

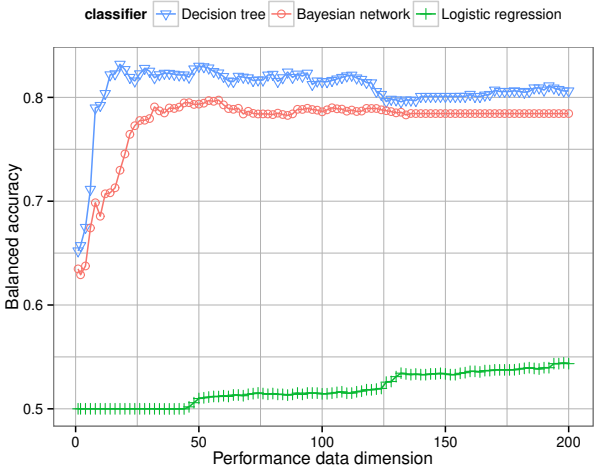

Figure 6: **Balanced accuracy as a function of data dimension**

accuracy becomes better for logistic classifier but shows marginal benefits for the other classifiers.

Figure 5 also shows that when combining grouping with undersampling, the balanced accuracy improves. This indicates that grouping is compatible with resampling technique.

6.6 Dimensionality Reduction

Data dimensionality affects both efficiency and accuracy for a learning task. Higher dimensions in general improves accuracy but may also hit the "curse of dimensionality"that hurts accuracy when dimensionality becomes very high [9]. Efficiency-wise, higher dimension slows down classification.

Figure 6 shows the accuracy for different data dimensionality. We can see that for decision trees and Bayesian network, the accuracy stays relatively stable as dimensionality decreases from 200 to 25 and drops significantly afterwards. The accuracy of logistic classifier decreases slightly as the dimensionality is reduced.

Figure 7 shows the training time impact of dimensionality reduction. Overall, reducing dimension significantly helps training time. For example, it takes more than 45 minutes to learn a decision tree classifier with 200 dimension data but takes less than 5 minutes for 25 dimensions. The training time of decision tree and Bayesian network is almost linear of data dimension. Logistic regression classifier has jittering training time but follows roughly similar trend. Considering Figure 6 and that the dimensionality reduction process takes around 2 minutes at most, we can exploit dimensionality reduction to a large extent for efficiency benefits without severely hurting accuracy.

6.7 Accounting for Baseline

Classifier	Balanced accuracy	False positive	False negative
Sim-Logistic	0.69	379	215
Conc-Logistic	0.61	99	290
Sub-Logistic	0.55	45	334
Conc-DTree	0.81	72	138
Sub-DTree	0.72	148	204
Con-TAN	0.79	411	142
Sub-TAN	0.69	563	209

Table 8: Result of including baseline in classification with concatenation approach, subtraction approach and similarity logistic learning.

To leverage the baseline vectors in comparative analysis, we explore three approaches: concatenating baseline and target vectors; subtracting baseline from target; and the similarity learning classifier (§4.5).

Table 8 shows the result of these three approaches. Concatenation has higher accuracy than subtraction for all three classifiers. But it doubles the dimensionality. Subtraction has lower accuracy but preserves data dimensionality. Our similarity logistic classifier achieves higher accuracy than the concatenation and subtraction logistic classifiers. It also has lower training time than than concatenation logistic classifier.

As explained in §6.2, training on pairs of performance

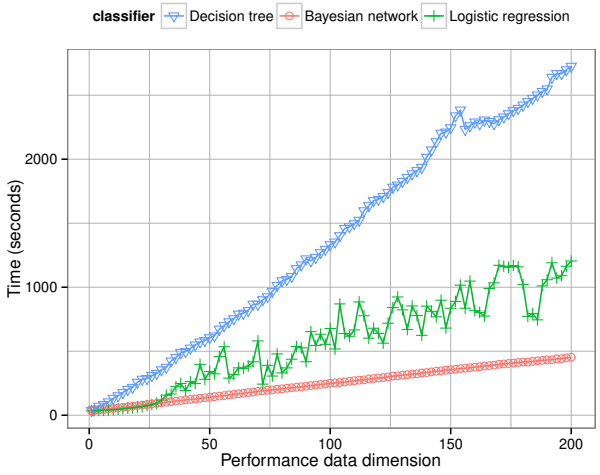

Figure 7: **Training time as a function of data dimension.**

vectors reduces the dataset so we cannot quantify the accuracy changes from not using baseline. However, looking at the false negative rates, concatenation decision tree, logistic and similarity logistic classifiers achieve lower rates than Table 6, which signals potential benefit of including baseline data.

6.8 Efficiency

Our original motivation for an automated analysis solution is to improve efficiency. Under the new learning scheme (grouped data, undersampling and cost adjustment), without reducing dimensions, the training time of logistic regression, decision tree and Bayesian network classifiers are on average 35, 5 and 3 minutes respectively, which enables the solution to be adopted in continuous performance testing.

Notice that compared with training with the collected data, even though partitioning the data and learning a group of classifiers increase the number of subtasks, the total training time actually decreases for decision tree and Bayesian network. This means some subtasks achieve much more than linear reduction in training time due to the reduced number of training examples.

7 Related Work

Recent research seeks to automate the comparative performance analysis through three complimentary directions: statistical technique like control charts [23], association rule learning [14], and classification learning [11]. We explored using pure statistical techniques in our early prototype but found the assumptions of these techniques such as each performance metric follows a Gaussian distribution do not hold in our large-volume performance dataset. Also our 200-dimension performance metrics shows weak association rules that result in low accuracy if used as anomaly detection criteria. Our work takes the classification learning approach similar to Cohen et al. [11] and uses its proposed TAN (Tree-Augmented Bayesian Networks) model.

Unlike [11], our work focuses on the pragmatic experience in building the model and incorporating other state-of-the-art classifiers into an analysis engine for a commercial system. In particular, we discuss five important data issues that are not addressed in previous work. These issues cause high false negative rates which prevent the classifiers from being adopted. We show that addressing these issues achieves significant false negative rate reduction and improves accuracy to acceptable range.

Jain's book on general performance analysis [17] provides a comprehensive guide on the statistical techniques useful for analyzing system performance data. This guide is popularly referred among performance analysts in practice. Our work automatically induces classification models by learning from the past wisdom of performance analysts.

There are other works on performance signature construction [8, 12], modeling [6, 25, 26, 28], issues detection and diagnosis [4, 5, 10, 15, 18, 19, 22]. Our problem domain is comparative analysis of performance data as an oracle for performance testing. Therefore it is orthogonal to the above directions.

8 Conclusion

Comparative performance analysis is commonly used to analyze performance measurement data for evolving systems. The efficiency of the analysis affects whether performance testing can be run continuously in a timely fashion. The accuracy of the analysis affects the quality control of system performance. Currently, this analysis is mainly conducted by human analysts, which is time consuming and can be ad-hoc.

In this paper, we share our experience in building an automatic analysis engine for a commercial system from a leading data warehouse company. We discuss and address several practical issues overlooked in previous literature that cause an average of 86% false negative rate in the initial evaluation of state-of-the-art classifiers. We show that resolving these issues and leveraging the data properties can help reduce false negative rate to as low as 16% and achieve maximum 0.91 balanced accuracy, a 0.3 improvement. The training time of the engine can be as low as 3 minutes, which meets our efficiency goal.

References

[1] Btrfs did regress hard in the Linux 2.6.35 kernel. `http://www.phoronix.com/scan.php?page=news_item&px=ODQ4Nw`.

[2] Chromium performance testing framework, Telemetry. `http://www.chromium.org/developers/telemetry`.

[3] Mozilla performance testing framework, Talos. `https://wiki.mozilla.org/Buildbot/Talos`.

[4] AGUILERA, M. K., MOGUL, J. C., WIENER, J. L., REYNOLDS, P., AND MUTHITACHAROEN, A. Performance debugging for distributed systems of black boxes. In *SOSP '03* (New York, NY, USA, 2003), ACM, pp. 74–89.

[5] ATTARIYAN, M., CHOW, M., AND FLINN, J. X-ray: automating root-cause diagnosis of performance anomalies in production software. In *OSDI'12* (Berkeley, CA, USA, 2012), USENIX Association, pp. 307–320.

[6] BARHAM, P., DONNELLY, A., ISAACS, R., AND MORTIER, R. Using magpie for request extraction and workload modelling. In *OSDI'04* (Berkeley, CA, USA, 2004), USENIX Association, pp. 18–18.

[7] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] BODÍK, P., GOLDSZMIDT, M., AND FOX, A. Hilighter: automatically building robust signatures of performance behavior for small- and large-scale systems. In *SysML'08* (Berkeley, CA, USA, 2008), USENIX Association, pp. 3–3.

[9] CARUANA, R., KARAMPATZIAKIS, N., AND YESSENALINA, A. An empirical evaluation of supervised learning in high dimensions. In *ICML '08* (New York, NY, USA, 2008), ACM, pp. 96–103.

[10] CHEN, T., ANANIEV, L. I., AND TIKHONOV, A. V. Keeping kernel performance from regressions. In *OLS' 07* (2007), vol. 1, pp. 93–102.

[11] COHEN, I., GOLDSZMIDT, M., KELLY, T., SYMONS, J., AND CHASE, J. S. Correlating instrumentation data to system states: a building block for automated diagnosis and control. In *OSDI'04* (Berkeley, CA, USA, 2004), USENIX Association, pp. 16–16.

[12] COHEN, I., ZHANG, S., GOLDSZMIDT, M., SYMONS, J., KELLY, T., AND FOX, A. Capturing, indexing, clustering, and retrieving system history. In *SOSP '05* (New York, NY, USA, 2005), ACM, pp. 105–118.

[13] DOMINGOS, P. Metacost: A general method for making classifiers cost-sensitive. In *KDD '99* (New York, NY, USA, 1999), ACM, pp. 155–164.

[14] FOO, K. C., JIANG, Z. M., ADAMS, B., HASSAN, A. E., ZOU, Y., AND FLORA, P. Mining performance regression testing repositories for automated performance analysis. In *QSIC '10* (Washington, DC, USA, 2010), IEEE Computer Society, pp. 32–41.

[15] GRECHANIK, M., FU, C., AND XIE, Q. Automatically finding performance problems with feedback-directed learning software testing. In *ICSE '12* (Piscataway, NJ, USA, 2012), IEEE Press, pp. 156–166.

[16] GUNTHER, N. *The Practical Performance Analyst*. McGraw-Hill series on computer communications. iUniverse, 2000.

[17] JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley, 1991.

[18] JIN, G., SONG, L., SHI, X., SCHERPELZ, J., AND LU, S. Understanding and detecting real-world performance bugs. In *PLDI '12* (New York, NY, USA, 2012), ACM, pp. 77–88.

[19] KILLIAN, C., NAGARAJ, K., PERVEZ, S., BRAUD, R., ANDERSON, J. W., AND JHALA, R. Finding latent performance bugs in systems implementations. In *FSE '10* (New York, NY, USA, 2010), ACM, pp. 17–26.

[20] MASNADI-SHIRAZI, H., AND VASCONCELOS, N. Risk minimization, probability elicitation, and cost-sensitive SVMs. In *ICML '10* (Haifa, Israel, June 2010), J. Fürnkranz and T. Joachims, Eds., Omnipress, pp. 759–766.

[21] MYTKOWICZ, T., DIWAN, A., HAUSWIRTH, M., AND SWEENEY, P. F. Producing wrong data without doing anything obviously wrong! In *ASPLOS XIV* (New York, NY, USA, 2009), ACM, pp. 265–276.

[22] NAGARAJ, K., KILLIAN, C., AND NEVILLE, J. Structured comparative analysis of systems logs to diagnose performance problems. In *NSDI '12* (Berkeley, CA, USA, 2012), USENIX Association, pp. 26–26.

[23] NGUYEN, T. H., ADAMS, B., JIANG, Z. M., HASSAN, A. E., NASSER, M., AND FLORA, P. Automated detection of performance regressions using statistical process control techniques. In *ICPE '12* (New York, NY, USA, 2012), ACM, pp. 299–310.

[24] PROVOST, F. Machine learning from imbalanced data sets 101 (extended abstract).

[25] SHEN, K. Request behavior variations. In *ASPLOS XV* (New York, NY, USA, 2010), ACM, pp. 103–116.

[26] SHEN, K., ZHONG, M., AND LI, C. I/O system performance debugging using model-driven anomaly characterization. In *FAST '05* (Berkeley, CA, USA, 2005), USENIX Association, pp. 23–23.

[27] SMITH, J. E. Characterizing computer performance with a single number. *Commun. ACM 31*, 10 (oct 1988), 1202–1206.

[28] STEWART, C., AND SHEN, K. Performance modeling and system management for multi-component online services. In *NSDI '05* (Berkeley, CA, USA, 2005), USENIX Association, pp. 71–84.

[29] WITTEN, I. H., FRANK, E., AND HALL, M. A. *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.