Opprentice: Towards Practical and Automatic Anomaly Detection Through Machine Learning

Dapeng Liu[†], Youjian Zhao[†], Haowen Xu[†], Yongqian Sun[†], Dan Pei[†][*], Jiao Luo[‡],
Xiaowei Jing[§], Mei Feng[§]
[†]Tsinghua University, [‡]Baidu, [§]PetroChina
[†]Tsinghua National Laboratory for Information Science and Technology
{liudp10, xhw11, sunyq12}@mails.tsinghua.edu.cn, {zhaoyoujian, peidan}@tsinghua.edu.cn, luojiao01@baidu.com, {jxw, fm}@petrochina.com.cn

ABSTRACT

Closely monitoring service performance and detecting anomalies are critical for Internet-based services. However, even though dozens of anomaly detectors have been proposed over the years, deploying them to a given service remains a great challenge, requiring manually and iteratively tuning detector parameters and thresholds. This paper tackles this challenge through a novel approach based on supervised machine learning. With our proposed system, Opprentice (**Op**erators' ap**prentice**), operators' only manual work is to periodically label the anomalies in the performance data with a convenient tool. Multiple existing detectors are applied to the performance data in parallel to extract anomaly features. Then the features and the labels are used to train a random forest classifier to automatically select the appropriate detector-parameter combinations and the thresholds. For three different service KPIs in a top global search engine, Opprentice can *automatically* satisfy or approximate a reasonable accuracy preference (recall $\geq$ 0.66 and precision $\geq$ 0.66). More importantly, Opprentice allows operators to label data in only tens of minutes, while operators traditionally have to spend more than ten days selecting and tuning detectors, which may still turn out not to work in the end.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring

General Terms

Measurement; Design

Keywords

Anomaly Detection; Tuning Detectors; Machine Learning

1. INTRODUCTION

Closely monitoring service quality and detecting performance anomalies are critical for Internet-based services, such as search engines, online shopping, and social networking. For example, a search engine can monitor its search response time (SRT) [1] by detecting anomalies in SRT time series in an online fashion. However, even though dozens of anomaly detectors [1–24] have been proposed over the years in the context of both Internet-based services and telecom services, deploying detectors to a given service remains a great challenge. This is, we argue, because there exists no convenient method to automatically match operators' practical detection requirements with the capabilities of different detectors. Currently, for a given service quality metric, selecting and applying detectors usually require manually and iteratively tuning the internal parameters of detectors and the detection thresholds, which operators are neither interested nor feel comfortable in doing. Instead, based on our experience of working with operators from a large search engine, a campus network, and an enterprise network, operators are used to specify simple requirements for detection accuracy and manually spot-check anomalies occasionally. As a result, services either settle with simple static thresholds (*e.g.*, Amazon Cloud Watch Alarms [24]), intuitive to operators although unsatisfying in detection performance, or, after time-consuming manual tuning by algorithm designers, end up with a detector specifically tailored for the given service, which might not be directly applicable to other services.

We elaborate on the challenges of anomaly detection by reviewing the current practice. The detectors proposed in the literature [1–24] typically work on (*time*, value) pair time series data,[1] which we call KPI (Key Performance Indicator) data hereinafter. Given a KPI, the first step for the anomaly detection practitioner is to collect the requirements from the service operators. This step encounters **Definition Challenges**: it is difficult to precisely define anomalies in reality [21, 25]. In particular, the operators often have trouble describing their domain knowledge completely when "put on the spot" [25]. In addition, it is often impossible for the operators to quantitatively define anomalies, let alone to translate the vague definition into specific parameters and thresholds of a detector [21] (*e.g.*, how many times of the standard deviation [1]) . On the contrary, according to our experience of building detection systems with operators, they prefer to describe the anomalies qualitatively, with anecdotal anomaly cases as examples.

[*] Dan Pei is the corresponding author.

[1] In this paper, we focus on the performance anomalies of time series (also known as volume-based anomalies) rather than other types of anomalies, *e.g.*, intrusion detection using the payload information of packets.

IMC'15, October 28–30, 2015, Tokyo, Japan.
© 2015 ACM. ISBN 978-1-4503-3848-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2815675.2815679.

Detector Challenges: In order to provide a reasonable detection accuracy, selecting the most suitable detector requires both the algorithm expertise and the domain knowledge about the given service KPI. The best parameters and thresholds of a given detector often highly depend on the actual data in the studied service. As such, it is very time-consuming to tune the parameters and thresholds of the detector. Sometimes, it might even require a combination of multiple detectors [8, 21, 22] for a given KPI. As a result, many rounds of time-consuming iterations between anomaly detection practitioners and operators are needed to find appropriate detectors and tune their parameters and thresholds. In reality, we observe that it is not uncommon for operators to give up after a few rounds of iterations and settle with static threshold-based detection.

To address the definition challenge and the detector challenge, we advocate for using supervised machine learning techniques,[2] well known for being able to capture complex concepts based on the features and the labels from the data (*e.g.*, KPI data). Our approach relies on two key observations. First, it is straightforward for operators to visually inspect the time series data and label anomaly cases they identified [1, 4, 9, 12, 14, 17, 26]. Operators can periodically (*e.g.*, weekly) label the cases as new data arrive, the only manual work for operators. Because anomalies are typically infrequent [16], the time for labeling is quite reasonable with the help of a dedicated labeling tool, as shown in [27] and §4.2. The second key observation is that the anomaly severities measured by different detectors can naturally serve as the features in machine learning, so each detector can serve as a feature extractor (see §4.3). Opprentice then learns from the labeled data, automatically capturing the domain knowledge from the operators, just as a smart and diligent apprentice of the operators would do. Specifically, multiple detectors are applied to the KPI data in parallel to extract features. Then the features and the labels are used to train a machine learning model, *i.e.*, random forests [28], to automatically select the appropriate detector-parameter combinations and the thresholds. The training objective is to maximally satisfy the operators' accuracy preference.

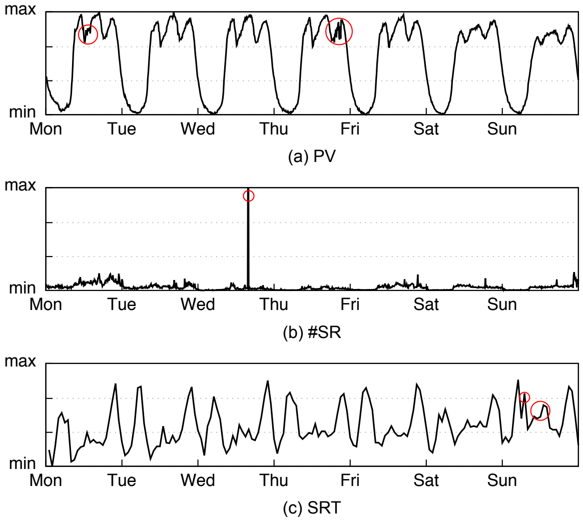
The major contributions of the paper are summarized as follows. First, to the best of our knowledge, Opprentice is the first detection framework to apply machine learning to acquiring realistic anomaly definitions and automatically combining and tuning diverse detectors to satisfy operators' accuracy preference. Second, Opprentice addresses a few challenges in applying machine learning to such a problem: labeling overhead, infrequent anomalies, class imbalance, and irrelevant and redundant features, elaborated on in §3.2 and addressed in §4. Third, we build and evaluate Opprentice in a top global search engine (§5). 14 existing detectors have been implemented and plugged into Opprentice. For three different service KPIs, Opprentice can *automatically* satisfy or approximate a reasonable accuracy preference (recall $\geq$ 0.66 and precision $\geq$ 0.66). Furthermore, although the best performing detector-parameter combination changes with different KPIs, Opprentice consistently performs similarly or even better than them. More importantly, Opprentice takes operators only tens of minutes to label data. In comparison, operators traditionally have to spend days learning and selecting detectors, then another tens of days tuning them, which may still turn out not to work in the end. We believe this is the first anomaly detection framework that does not require manual detector selection, parameter configuration, or threshold tuning.

2. BACKGROUND

In this section, we first introduce the background of KPI anomaly detection. Then we present the goals and the challenges of designing Opprentice.

2.1 KPIs and KPI Anomalies

KPIs: The KPI data, which Opprentice aims to work with, are the time series data with the format of (timestamp, value). These data can be collected from SNMP, syslogs, network traces, web access logs, and other data sources. In this paper, we choose three representative KPIs from a global top search engine as a case study. Table 1 describes their basic information, and Fig. 1 shows their 1-week examples. We hide the absolute values for confidentiality. Fig. 1(a) shows the search page view (PV), which is the number of successfully served queries. PV has a significant influence on the revenue of the search engine. Fig. 1(b) shows the number of slow responses of search data centers (#SR), which is an important performance metric of the data centers. Fig. 1(c) is the the 80th percentile of search response time (SRT). This KPI has a measurable impact on the users' search experience [29].



(a) PV

(b) #SR

(c) SRT

Figure 1: 1-week examples of three major KPIs of the search engine. The circles mark some obvious (not all) anomalies.

Beyond the physical meanings, the characteristics of these KPI data are also different. First, they have different levels of seasonality. For example, by visually inspecting, we see that the PV is much more regular than the other two KPIs and shows a strong seasonality. In addition, the dispersions of the KPIs are different too. Since we have to hide the absolute values, we use the coefficient of variation (C_v) to measure the dispersions. C_v equals the standard deviation divided by the mean. In Table 1, #SR has $C_v = 209\%$ and is spread out the most; SRT has $C_v = 7\%$ and concentrates the most to the mean.

Table 1: Three kinds of KPI data from the search engine.

	PV	#SR	SRT
Interval (minute)	1	1	60
Length (week)	25	19	16
Seasonality	Strong	Weak	Moderate
C_v	0.48	2.1	0.07

[2] In the rest of this paper, machine learning refers particularly to supervised machine learning.

Anomalies: KPI time series data can also present several unexpected patterns (*e.g.*, jitters, slow ramp-ups, sudden spikes and dips) in different severity levels, such as a sudden drop by 20% or 50%. When identifying anomalies, operators care about certain patterns with different severities, which can vary among KPIs. Fig. 1 shows a few anomaly examples. However, this knowledge is difficult to be described accurately by some pre-defined rules [1,2]. This is because operators usually determine anomalies according to their own understandings of the KPIs and the real operational demands. Throughout this paper, we assume that operators have no concept drift [30] regarding anomalies. This is consistent with what we observed when the operators labeled months of data studied in this paper.

To identify anomalies automatically, researchers have proposed many detectors using a variety of techniques. We call them *basic detectors* in the rest of the paper. More details about detectors will be discussed in §4.3.

2.2 Problem and Goal

The KPI data labeled by operators are the so called "ground truth". The fundamental goal of anomaly detection is to be accurate, *e.g.*, identifying more anomalies in the ground truth, and avoiding false alarms. We use *recall* ($\frac{\text{\# of true anomalous points detected}}{\text{\# of true anomalous points}}$) and *precision* ($\frac{\text{\# of true anomalous points detected}}{\text{\# of anomalous points detected}}$) to measure the detection accuracy. Precision describes what matters to operators better than false positive rate (FPR), because anomalies are infrequent [31]. Precision is also equivalent to 1-FDR (false discovery rate: $\frac{\text{\# of false anomalous points detected}}{\text{\# of anomalous points detected}}$). Based on our experience, operators not only understand the concepts of recall and precision, but can also specify their accuracy preference using them in the format of "recall $\geq x$ and precision $\geq y$ ". For example, the operators we worked with specified "recall ≥ 0.66 and precision ≥ 0.66" as the accuracy preference, which is considered as the **quantitative goal** of Opprentice in this paper. These values come from the operators' experience of using other detectors and their accuracy before. As anomalies are relatively few in the data, it is difficult for those detectors to achieve both high recall and precision. In fact, precision and recall are often conflicting. The trade-off between them is often adjusted according to real demands. For example, busy operators are more sensitive to precision, as they do not want to be frequently disturbed by many false alarms. On the other hand, operators would care more about recall if a KPI, *e.g.*, revenue, is critical, even at the cost of a little lower precision. We also evaluate Opprentice under different accuracy preference in §5.

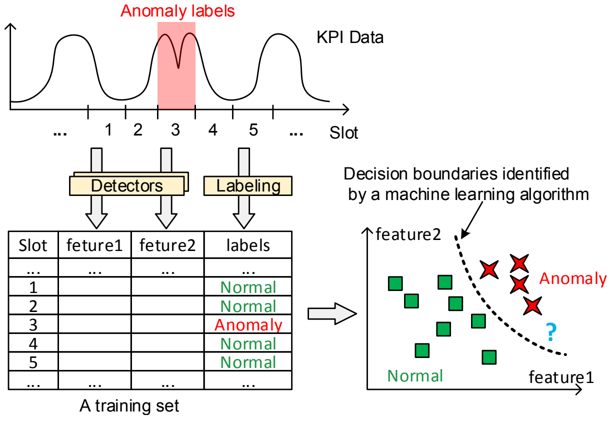
In addition to the above quantitative goal of accuracy, Opprentice has one **qualitative goal**: being automatic enough so that the operators would not be involved in selecting and combining suitable detectors, or tuning them.

In this paper, we focus on identifying anomalous behaviors in KPI time series. This is an important first step for monitoring the service performance. However, further investigation and troubleshooting of those anomalies are beyond the research scope of this paper.

3. OPPRENTICE OVERVIEW

3.1 Core Ideas

Opprentice approaches the above problem through supervised machine learning. Supervised machine learning can be used to automatically build a classification model from historical data, and then classify future data based on the model. Because of the data-driven property, supervised machine learning has become a popular solution where hand-crafted rules of classification are difficult to specify in advance, *e.g.*, computer vision and data mining. In anomaly detection, manually pre-defining anomalies is also challenging, which motivates us to tackle the problem through supervised machine learning.



Slot	feture1	feture2	labels
...	...	...	...
1			Normal
2			Normal
3			Anomaly
4			Normal
5			Normal
...	...	...	...

Figure 2: High level idea of applying machine learning in Opprentice.

Fig. 2 shows the high-level idea of how machine learning is applied in Opprentice. We generate a training set from historical KPI data, which is used by a machine learning algorithm to build a classification model. To this end, first, operators need to label the anomalies in the data. In the meanwhile, we use existing basic anomaly detectors to quantify anomalous level of the data from their own perspectives, respectively. The results of the detectors are used as the features of the data. The features and operators' labels together form the training set. Then a machine learning algorithm takes advantage of a certain technique to build a classification model. For example, given the decision boundaries in Fig. 2, the point represented by the question mark is classified as an anomaly.

In this way, operators' only job in building an anomaly detection system is to label anomaly cases, which is much easier (§4.2) and costs less time (§5.7). The rest would be handled by the machine learning based framework, including combining diverse detectors and adjusting their thresholds. Note that, although prior work has applied machine learning algorithms to anomaly detection [16,20,32], they only deem machine learning algorithms as *basic detectors*. To the best of our knowledge, Opprentice is the first framework that use machine learning to automatically *combine* and *tune* existing detectors to satisfy operators' detection requirements (anomaly definitions and the detection accuracy preference). Furthermore, to the best of our knowledge, this is the first time that different detectors are modeled as the feature extractors in machine learning (§4.3).

3.2 Addressing Challenges in Machine Learning

Although the above machine learning based idea seems promising, applying it in designing Opprentice poses a number of interesting and practical challenges.

- **Labeling overhead.** Labeling anomalies requires a lot of manual efforts. To help operators label effectively, we developed a dedicated labeling tool with a simple and convenient interaction interface. §5.7 shows that labeling time of our studied KPIs with our tool is less than 6 minutes for each month of data.

- **Incomplete anomaly cases.** The performance of machine learning algorithms can be affected by whether the training set contains enough anomaly cases. However, since anomalies occur less frequently in practice, an arbitrary training set is unlikely to cover enough anomalies [16]. For example, new types of anomalies might emerge in the future. To address this challenge, we incrementally retrain the classifier with newly labeled data. Through this way, Opprentice is able to catch and learn new types of anomalies that do not show up in the initial training set.

- **Class imbalance problem.** Another effect of infrequent anomalies is that the normal data always outnumber the anomalies in the training set. When learning from such "imbalanced data", the classifier is biased towards the large (normal) class and ignores the small (anomaly) class [31]. It results in low detection rate, which may not satisfy operators' accuracy preference. We solve this problem in §4.5 through adjusting the machine learning classification threshold (**cThld** henceforth).

- **Irrelevant and redundant features.** To save manual efforts, we neither select the most suitable detectors nor tune their internal parameters. Instead, many detectors with different parameters are used simultaneously to extract features (§4.3). In this case, some of the features would be either irrelevant to the anomalies or redundant with each other. Prior work has demonstrated that some learning algorithms would degrade in accuracy when handling such features. We solve this problem by using an ensemble learning algorithm, *i.e.*, random forests [28], which is relatively robust and works well for our problem (§5).

4. DESIGN

In this section we first present the architecture of Opprentice, and then describe the design details.

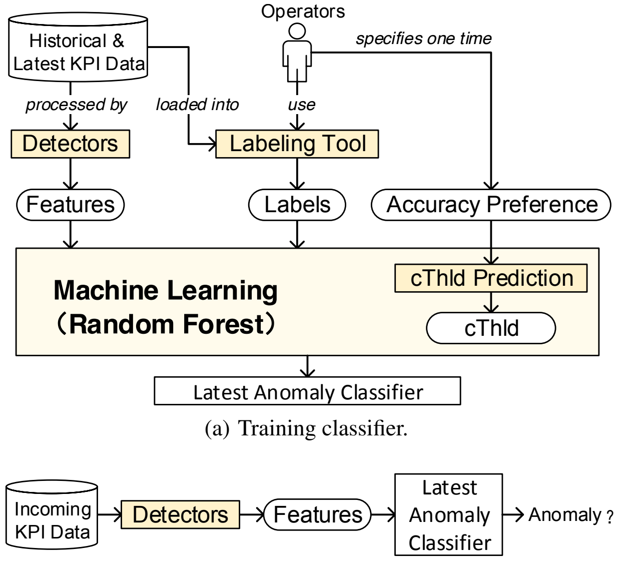
4.1 Architecture

Fig. 3 illustrates the architecture of Opprentice. From the operators' view, they interact with Opprentice in two ways (Fig. 3(a)). First, before the system starts up, operators specify an accuracy preference (recall $\geq x$ and precision $\geq y$), which we assume does not change in this paper. This preference is later used to guide the automatic adjustment of the cThld. Second, the operators use a convenient tool to label anomalies in the historical data at the beginning and label the incoming data periodically (*e.g.*, weekly). All the data are labeled only once.

From the Opprentice-side view, first in Fig. 3(a), an anomaly classifier is trained as follows. Numerous detectors function as feature extractors for the data. Based on the features together with the operators' labels, a machine learning algorithm (*e.g.*, random forests used in this paper) incrementally retrains the anomaly classifier with both the historical and the latest labeled data. After that, in Fig. 3(b), the same set of detectors extract the features of incoming data, and the classifier is used to detect/classify them. Note that, unlike the traditional way, the detectors here only extract features rather than reporting anomalies by themselves. Next, we introduce the design of each component in detail.

4.2 Labeling Tool

We developed a dedicated tool to help operators effectively label anomalies in historical data. The user interface of the tool is shown in the left part of Fig. 4, with a brief user manual on the right side. The tool works as follows. First, it loads KPI data, and displays them with a line graph in the top panel. To assist operators in identifying anomalies, the data of the last day and the last week are also shown in light colors. The operators can use the arrow keys on the keyboard to navigate (forward, backward, zoom in and zoom out) through the data. Once the operators have identified anomalies, they can left click and drag the mouse to label the window of anomalies, or right click and drag to (partially) cancel previously labeled window. Besides, they can adjust the Y axis scale via the slider on the right side. The data navigator in the bottom panel shows a zoom-out view of the data.



(a) Training classifier.

(b) Detecting anomaly.

Figure 3: Opprentice architecture.

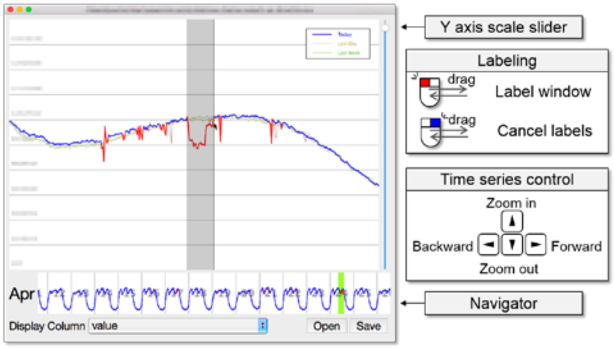


Figure 4: Labeling tool.

The labeling tool is effective because operators do not have to label each time bin one by one. They first see a relatively zoomed-out view of the KPI curve. In this view, we do not smooth the curve. Thus, even if one time bin is anomalous, it is visible to operators. Then, operators can zoom in to locate the specific anomalous time bin(s), and label them by a window. Labeling windows, as opposite to individual time bins, significantly reduces labeling overhead. §5.7 shows that it only takes operators a few minutes to label a month of data in our studied KPIs.

One issue of labeling is that errors can be introduced, especially that the boundaries of an anomalous window are often extended or narrowed when labeling. However, machine learning is well known for being robust to noises. Our evaluation in §5 also attests that the real labels of operators are viable for learning.

Our labeling tool in spirit is similar to WebClass [27], a labeling tool for NetFlow data. However, WebClass cannot be used directly in our problem because it only supports NetFlow data rather than general time series data. More importantly, it only allows operators to label the anomalies already identified by detectors as *false positives* or *unknown*. In contrast, our labeling tool enables operators to freely label all the data rather than labeling the detection results.

4.3 Detectors

We now describe how detectors function as extractors of anomaly features in Opprentice, and introduce the considerations when choosing detectors to work with Opprentice.

4.3.1 Detectors As Feature Extractors

Inspired by [21, 33], we represent different detectors with a unified model:

$$\text{data point} \xrightarrow{\text{a detector with parameters}} \text{severity} \xrightarrow{\text{sThld}} \{1, 0\}$$

First, when a detector receives an incoming data point, it internally produces a non-negative value, called *severity*, to measure how anomalous that point is. For example, Holt-Winters [6] uses the residual error (*i.e.*, the absolute difference between the actual value and the forecast value of each data point) to measure the severity; historical average [5] assumes the data follow Gaussian distribution, and uses how many times of standard deviation the point is away from the mean as the severity. Most detectors are parameterized and have a set of *internal parameters*. For example, Holt-Winters has three parameters $\{\alpha, \beta, \gamma\}$, and historical average has one parameter of window length. The severity of a given data point depends on both the detector and its internal parameters. Afterwards, a detector further needs a threshold to translate the severity into a binary output, *i.e.*, anomaly (1) or not (0). We call this threshold the severity threshold (**sThld** henceforth).

Since the severity describes the anomalous level of data, it is natural to deem the severity as the anomaly feature. To produce features, for each parameterized detector, we sample their parameters [34] so that we can obtain several fixed detectors. We call a detector with specific sampled parameters a (detector) *configuration*. Thus a configuration acts as a *feature extractor*:

$$\text{data point} \xrightarrow{\text{configuration (detector + sampled parameters)}} \text{feature}$$

The feature extraction, training, and classification (detection) in Opprentice are all designed to work with individual data points, not anomalous windows. This way, the machine learning algorithm can have enough training data, and the classifier can detect individual anomalous data point fast.

4.3.2 Choosing Detectors

When choosing detectors, we have two general requirements. First, the detectors should fit the above model, or they should be able to measure the severities of data. In fact, a lot of widely-used detectors all work in this way [1, 4–7, 10–12, 24]. Second, since anomalies should be detected timely, we require that the detectors can be implemented in an online fashion. This requires that once a data point arrives, its severity should be calculated by the detectors without waiting for any subsequent data. In addition, the calculation time should be less than the data interval, which is not difficult to fulfill. For example, the shortest data interval is one minute in our studied data. Besides, some detectors, such as those based on moving averages, need one window of data to warm up. We cope with such detectors by skipping the detection of the data in the warm-up window, which has no influence on the detection of future data.

Since we intend to free operators from carefully selecting detectors, the detectors meeting the above requirements are used to work with Opprentice without carefully evaluating their effectiveness. Although some detectors might be inaccurate in detecting certain KPIs (§5.3.1), Opprentice can find suitable ones from broadly selected detectors, and achieve a relatively high accuracy. In this paper, we implement 14 widely-used detectors (introduced later in §5.2) in Opprentice as a case study.

4.3.3 Sampling Parameters

We have two strategies to sample the parameters of detectors. The first one is to sweep the parameter space. We observe that the parameters of some detectors have intuitive meanings. For example, EWMA (Exponentially Weighted Moving Average) [11], a prediction based detector, has only one weight parameter $\alpha \in [0, 1]$. As α goes up, the prediction relies more upon the recent data than the historical data. Consequently, we can sample $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ to obtain 5 typical features from EWMA. As for the detectors with multiple parameters and a large parameter space, we can reduce the sampling granularity. For example, Holt-Winters has three [0,1] valued parameters α, β, and γ. To limit the number of samples, we can choose $\{0.2, 0.4, 0.6, 0.8\}$ for α, β, and γ, leading to $4^3 = 64$ features. Other types of detectors may need window parameters, and we can adopt windows of several points, days, or weeks according to the characteristics of the detectors. For example, moving average based detectors with a short window aim at identifying local anomalies, while time series decomposition [1] usually uses a window of weeks to capture long-term violations. Although such sampling strategies do not guarantee that we can find the most suitable parameters (or features) due to the relatively coarse sampling granularity, we only need a set of good enough features, and Opprentice can achieve a promising accuracy by combining them (§5).

On the other hand, the parameters of some complex detectors, *e.g.*, ARIMA (Autoregressive Integrated Moving Average) [10], can be less intuitive. Worse, their parameter spaces can be too large even for sampling. To deal with such detectors, we estimate their "best" parameters from the data, and generate only one set of parameters, or one configuration for each detector. The estimation method is specifically designed for each such detector. For example, [35,36] provide the parameter estimation for ARIMA. Besides, since the data characteristics can change over time, it is also necessary to update the parameter estimates periodically.

4.4 Machine Learning Algorithm

4.4.1 Considerations and Choices

We need to be careful when choosing machine learning algorithms. This is because in our problem, there are redundant and irrelevant features, caused by using detectors without careful evaluation. Some learning algorithms such as naive Bayes, logistic regression, decision tree, and linear SVM, will perform badly when coping with such training data (§5.3.2). Additionally, a promising algorithm should be less-parametric and insensitive to its parameters, so that Opprentice can be easily applied to different data sets. In this paper, we choose random forests [28], which has been proved to be robust to noisy features and work well in practice [28, 37]. Furthermore, random forests have only two parameters and are not very sensitive to them [38]. Our evaluation results also show that the random forest algorithm perform better, and are more stable than other commonly-used algorithms.

Note that we do understand that *feature selection* [39, 40] is a commonly used solution to mitigate the influences of irrelevant and redundant features. However, we do not explore feature selection in this paper and consider it as future work, because it could introduce extra computation overhead, and the random forest works well by itself (§5.3.2).

4.4.2 Random Forest

In the interest of space, we only introduce some basic ideas of random forests. More details are in [28].

Preliminaries: decision trees. A decision tree [41] is a popular learning algorithm as it is simple to understand and interpret. It has been used widely to uncover the relationships between features and labels [42, 43]. At a high level, it provides a tree model with various if-then rules to classify data. Fig. 5 shows a compacted decision tree learned from our SRT data set. The tree contains three if-then rules on the features of three detectors, *i.e.*, time series decomposition, singular value decomposition, and diff (See §5.2 for the details of the detectors). The numbers on branches, *e.g.*, 3 for time series decomposition, are the feature split points. The tree is then used to classify incoming data. The tree is greedily built top-down. At each level, it determines the *best* feature and its split point to separate the data into distinct classes as much as possible, or produce the "purest" sub nodes. A goodness function, *e.g.*, information gain and gini index, is used for quantifying such an ability of each feature. The tree grows in this way until every leaf node is pure (fully grown). In the decision tree, a feature is more important for classification if it is closer to the root. For example, in Fig. 5, the feature of time series decomposition is most effective to distinguish different data.

There are two major problems of decision tree. One is that the greedy feature selection at each step may not lead to a good final classifier; the other is that the fully grown tree is very sensitive to noisy data and features, and would not be general enough to classify future data, which is called *overfitting*. Some pruning solutions have been proposed to solve overfitting. For example, stop growing the tree earlier after it exceeds a threshold of depth. However, it is still quite tricky to determine such a threshold.

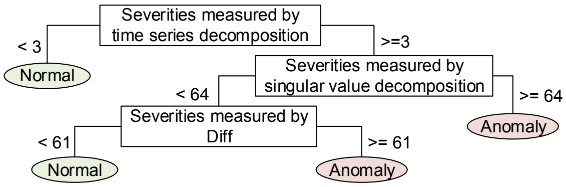

Severities measured by time series decomposition
< 3
>=3
Normal
Severities measured by singular value decomposition
< 64
>= 64
Anomaly
Severities measured by Diff
< 61
>= 61
Normal
Anomaly

Figure 5: Decision tree example. The rectangles represent the features extracted by different detectors and the ellipses are the classification results.

A Random forest is an ensemble classifier using many decision trees. Its main principle is that a group of weak learners (*e.g.*, individual decision trees) can together form a strong learner [44]. To grow different trees, a random forest adds some elements or randomness. First, each tree is trained on subsets sampled from the original training set. Second, instead of evaluating all the features at each level, the trees only consider a random subset of the features each time. As a result, some trees may be not or less affected by the irrelevant and redundant features if these features are not used by the trees. All the trees are fully grown in this way without pruning. The random forest then combines those trees by majority vote. That is, given a new data point, each of the trees gives its own classification. For example, if 40 trees out of 100 classify the point into an anomaly, its anomaly probability is 40%. By default, the random forest uses 50% as the classification threshold (*i.e.*, cThld).

The above properties of randomness and ensemble make random forests more robust to noises and perform better when faced with irrelevant and redundant features than decision trees.

4.5 Configuring cThlds

4.5.1 PC-Score: A Metric to Select Proper cThlds

We need to configure cThlds rather than using the default one (*e.g.*, 0.5) for two reasons. First, when faced with imbalanced data (anomalous data points are much less frequent than normal ones in data sets), machine learning algorithms typically fail to identify the anomalies (low recall) if using the default cThlds [31]. Second, operators have their own preference regarding the precision and recall of anomaly detection. Configuring cThlds is a general method to trade off between precision and recall [31]. In consequence, we should configure the cThld of random forests properly to satisfy the operators' preference.

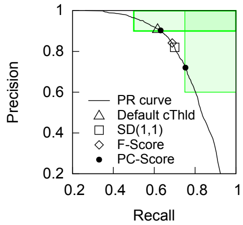


Figure 6: PR curve of a random forest trained and tested on the PV data. Different methods select different cThlds and result in different precision and recall. The two shaded rectangles represent two types of assumed operators' preferences.

Before describing how we configure the cThld, we first use Precision-Recall (PR) curves to provide some intuitions. PR curves is widely used to evaluate the accuracy of a binary classifier [45], especially when the data is imbalanced.[3] A PR curve plots precision against recall for every possible cThld of a machine learning algorithm (or for every sThld of a basic detector). Typically, there is a trade-off between precision and recall. Fig. 6 shows an exemplary PR curve derived from a random forest trained and tested on PV data. Two types of assumed operators' preferences (1) "recall $\geq$ 0.75 and precision $\geq$ 0.6" and (2) "recall $\geq$ 0.5 and precision $\geq$ 0.9" are represented by the shaded rectangles. Configuration of cThlds is to seek a proper point on the PR curve. In Fig. 6, the triangle symbol is selected by the default cThld 0.5. Besides, we also show the results of another two accurate metrics: a *F-Score* based method, which selects the point that maximizes F-Score $= \frac{2 \cdot precision \cdot recall}{precision + recall}$; *SD(1,1)*, a metric similar to that in [46], which selects the point with the shortest Euclidean distance to the upper right corner where the precision and the recall are both perfect. We see that in Fig. 6, the PR curve has points inside both the rectangles, however, the default threshold only satisfies the preference (2) but not (1); F-Score and SD(1,1) do not satisfy

[3] A similar method is Receiver Operator Characteristic (ROC) curves, which show the trade-off between the false positive rate (FPR) and the true positive rate (TPR). However, when dealing with highly imbalanced data sets, PR curves can provide a more informative representation of the performance [45].

either of the preferences. This is because these metrics select cThld without considering operators' preferences.

Motivated by the above fact, we develop a simple but effective accuracy metric based on F-Score, namely PC-Score (preference-centric score), to explicitly take operators' preference into account when deciding cThlds. First, for each point (r, p) on the PR curve, we calculate its PC-Score as follows:

$$\text{PC-Score}(r,p) = \begin{cases} \frac{2\cdot r\cdot p}{r+p}+1 & , r \geq R \text{ and } p \geq P \\ \frac{2\cdot r\cdot p}{r+p} & \text{, otherwise} \end{cases}$$

where R and P are from the operators' preference "recall $\geq R$ and precision $\geq P$". Basically, the PC-Score measures the F-Score of (r, p). In order to identify the point satisfying operators' preference (if one exists), we add an incentive constant of 1 to F-score if $r \leq R$ and $p \leq P$. Since F-Score is no more than 1, this incentive constant ensures that the points satisfying the preference must have the PC-Score larger than others that do not. Hence, we choose the cThld corresponding to the point with the largest PC-Score. In Fig. 6, we see that the two points selected based on the PC-Score are inside the two shaded recectangles, respectively. Note that, in the case when a PR curve has no points inside the preference region, the PC-Score cannot find the desired points, but it can still choose approximate recall and precision.

4.5.2 EWMA Based cThld Prediction

The above describes how to configure cThlds based on the PC-Score in an offline or "oracle" mode. That is, we configure cThlds after the data to be detected (also called a test set) have already arrived. These cThlds are the best ones we can configure for detecting those data, and are called *best cThlds*. However, in online detection, we need to predict cThlds for detecting future data.

To this end, an alternative method is k-fold cross-validation [47]. First, a historical training set is divided into k subsets of the same length. In each test (k tests in total), a classifier is trained using $k-1$ of the subsets and tested on the rest one with a cThld candidate. The candidate that achieves the smallest average PC-Score across the k tests is used for future detection. In this paper we use $k = 5$, and sweep the space of cThld with a very fine granularity of 0.001, that is, we evaluate 1000 cThld candidates in a range of $[0, 1]$.

However, we found that this cross-validation method does not work quite well in our problem (§5.6). A potential explanation is that, as shown in Fig. 7, the best cThlds can differ greatly over weeks. As a result, in the cross-validation, the cThld that achieves the highest average performance over all the historical data might not be similar to the best cThld of the future week.

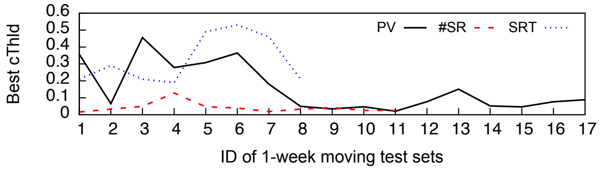


Figure 7: Best cThld of each week from the 9th week. The first 8-week data are used as the initial training set.

Our method is motivated by another observation in Fig. 7. That is, though the best cThlds changes over weeks, they can be more similar to the ones of the neighboring weeks. A possible explanation is that the underlying problems that cause KPI anomalies might last for some time before they are really fixed, so the neighboring weeks are more likely to have similar anomalies and require similar cThlds. Hence, we adopt EWMA [11] to predict the cThld of the i^{th} week (or the i^{th} test set) based on the historical *best cThlds*. Specifically, EWMA works as follows:

$$\text{cThld}_i^p = \begin{cases} \alpha \cdot \text{cThld}_{i-1}^b + (1-\alpha)\cdot \text{cThld}_{i-1}^p & , i > 1 \\ \text{5-fold prediction} & , i = 1 \end{cases}$$

cThld_{i-1}^b is the best cThld of the $(i-1)^{\text{th}}$ week. cThld_i^p is the predicted cThld of the i^{th} week, and also the one used for detecting the i^{th}-week data. $\alpha \in [0, 1]$ is the smoothing constant. For the first week, we use 5-fold cross-validation to initialize cThld_1^p. EWMA is simple but effective here as it does not require a lot of historical data to start. As α increases, EWMA gives the recent best cThlds more influences in the prediction. We use $\alpha = 0.8$ in this paper to quickly catch up with the cThld variation. Our results show that the EWMA based cThld prediction method gains a noticeable improvement when compared with the 5-fold cross-validation (§5.6).

5. EVALUATION

We implement Opprentice and 14 detectors with about 9500 lines of python, R, and C++ code. The machine learning block is based on the scikit-learn library [48]. In this section, we evaluate Opprentice using three kinds of KPI data from a top global search engine. These data are labeled by the operators using our labeling tool.

Fig. 8 shows the evaluation flow. In the first four steps, we compare the accuracy of each component of Opprentice with different approaches. The accuracy of Opprentice as a whole is shown in the §5.6. In addition to the accuracy, the qualitative goal of Opprentice, *i.e.*, being automatic, is also evaluated through directly applying Opprentice to three different KPIs without tuning. The only manual effort is to label the KPI data. We also interviewed the operators about their previous detector tuning time, and compare it with the labeling time (§5.7). Last, we evaluate the online detecting lag and the offline training time of Opprentice (§5.8). Next we first describe the data sets in §5.1 and the detectors we select in §5.2, then we show the evaluation results.

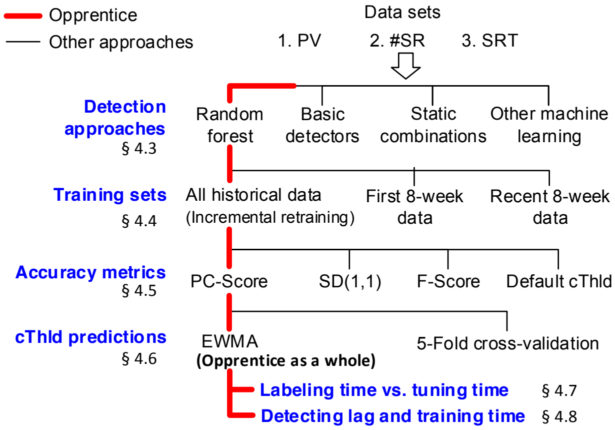


Figure 8: Evaluation flow.

5.1 Data sets

We collect three representative types of KPI data (*i.e.*, PV, #SR, and SRT) from a large search engine (§2.1). These data are labeled by the operators from the search engine using our labeling tool. There are 7.8%, 2.8%, and 7.4% data points are labeled as anomalies for PV, #SR, and SRT, respectively. Although the data we used are from the search engine, they are not special cases only for the search engine. For example, based on previous literature and our experience, the PV data we used are visually similar to other kinds of volume data. For example, the PV of other Web-based services [1, 49], the RTT (round trip time) [6] and the aggregated traffic volume [5] of an ISP, and online shopping revenue. So we believe that these three KPIs are sufficient to evaluate the idea of Opprentice, and we consider a more extensive evaluation with data from other domains beyond search as our future work. Table 2 shows several ways to generate training sets and test sets from the labeled data sets.

Table 2: Training sets and test sets. The test sets all start from the 9th week and move 1 week for each step. The training sets use different data before the test sets. I1 is the fashion of Opprentice (incremental retraining). Others are used for evaluating different strategies of training sets.

Training set	Test set	ID
All historical data	1-week moving window	I1
	4-week moving window	I4
Recent 8-week data	4-week moving window	R4
First 8-week data	4-week moving window	F4

5.2 Detector and Parameter Choices

According to the detector requirements (§4.3.2), in this proof of concept prototype, we evaluate Opprentice with 14 widely-used detectors (Table 3).

Table 3: Basic detectors and sampled parameters. Some abbreviations are MA (Moving Average), EWMA (Exponentially Weighted MA), TSD (Time Series Decomposition), SVD (Singular Value Decomposition), win(dow) and freq(uency).

Detector / # of configurations	Sampled parameters
Simple threshold [24] / 1	none
Diff / 3	last-slot, last-day, last-week
Simple MA [4] / 5	win = 10, 20, 30, 40, 50 points
Weighted MA [11] / 5	
MA of diff / 5	
EWMA [11] / 5	α = 0.1, 0.3, 0.5, 0.7, 0.9
TSD [1] / 5	win = 1, 2, 3, 4, 5 week(s)
TSD MAD / 5	
Historical average [5] / 5	
Historical MAD / 5	
Holt-Winters [6] / $4^3 = 64$	α, β, γ = 0.2, 0.4, 0.6, 0.8
SVD [7] / $5 \times 3 = 15$	row =10, 20, 30, 40, 50 points, column =3, 5, 7
Wavelet [12] / $3 \times 3 = 9$	win = 3, 5 ,7 days, freq = low, mid, high
ARIMA [10] / 1	Estimation from data
In total: 14 basic detectors / 133 configurations	

Two of the detectors were already used by the search engine we studied before this study. One is namely "Diff", which simply measures anomaly severities using the differences between the current point and the point of last slot, the point of last day, and the point of last week. The other one, namely "MA of diff", measures severities using the moving average of the difference between current point and the point of last slot. This detector is designed to discover continuous jitters. The other 12 detectors come from previous literatures. Among these detectors, there are two variants of detectors using MAD (Median Absolute Deviation) around the median, instead of the standard deviation around the mean, to measure anomaly severities. This patch can improve the robustness to missing data and outliers [3, 15]. In the interest of space, the details of these detectors and the ways they produce severities are not introduced further, but can be found in the references in Table 3. The sampled parameters of each detector are also shown in Table 3. Here, the parameter of ARIMA is estimated from the data. For other detectors, we sweep their parameter space.

In total, we have 14 detectors and 133 configurations, or 133 features for random forests. Note that, Opprentice is not limited to the detectors we used, and can incorporate emerging detectors, as long as they meet our detector requirements in §4.3.2.

5.3 Accuracy of Random Forests

Now we present the evaluation results. First, we compare the accuracy of random forests with other detection approaches in an offline mode. Since we are not aware of the thresholds of other approaches, we cannot compare specific recall and precision fairly. Alternatively, we use the area under the PR curve (AUCPR) [50] as the accuracy measure. The AUCPR is a single number summary of the detection performance over all the possible thresholds. The AUCPR ranges from 0 to 1. Intuitively, a detection approach with a large AUCPR is more likely to achieve high recall and precision.

5.3.1 Random Forests vs. Basic Detectors and Static Combinations of Basic Detectors

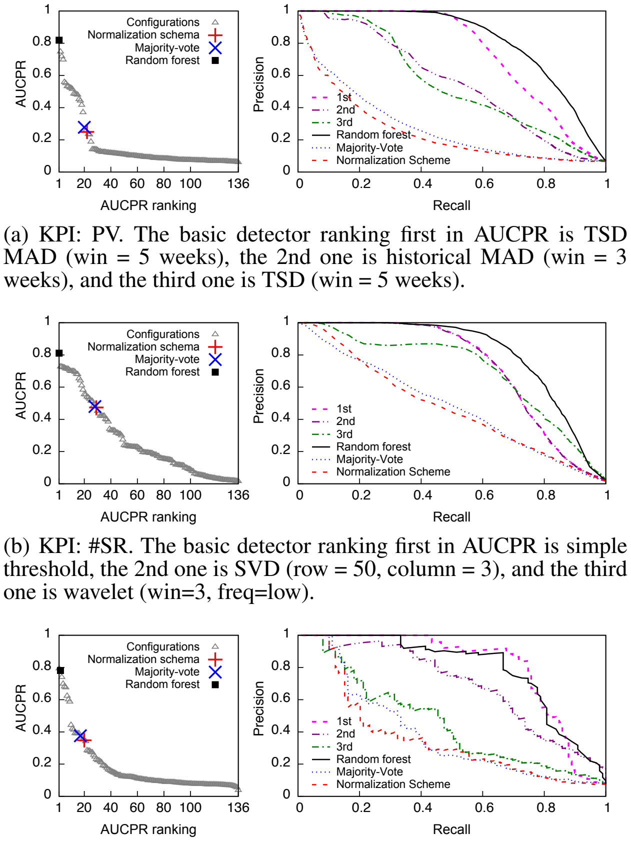
First, in Fig. 9, we would like to compare random forests with the 14 basic detectors with different parameter settings (133 configurations) in Table. 3. We also compare random forests with two static combination methods: the *normalization schema* [21] and the *majority-vote* [8]. These two methods are designed to combine different detectors, but they treat them equally no matter their accuracy. For comparison purposes, in this paper, we also use these two methods to combine the 133 configurations as random forests do. All the above approaches detect the data starting from the 9th week. The first 8 weeks are used as the initial training set for random forests.

Table 4: Maximum precision when recall $\geq$ 0.66. The precision greater than 0.66 is shown in bold. The top 3 basic detectors are different for each KPI (see Fig. 9 for their names).

Detection approach	Precision		
	PV	#SR	SRT
Random forest	**0.83**	**0.87**	**0.89**
Normalization scheme	0.11	0.30	0.21
Majority-vote	0.12	0.32	0.19
1st basic detector	**0.67**	**0.71**	**0.92**
2nd basic detector	0.39	**0.70**	0.61
3rd basic detector	0.37	**0.67**	0.24

Focusing on the left side of Fig. 9, we see that for the AUCPR, random forests rank the first in Fig. 9(a) and Fig. 9(b), and rank the second in Fig. 9(c), where the AUCPR of random forests is only 0.01 less than the highest one. On the other hand, the AUCPR of the two static combination methods is always ranked low. This is because most configurations are inaccurate (having very low AUCPR in Fig. 9), as we do not manually select proper detectors or tune their parameters. However, the two static combination methods treat all the configurations with the same priority (*e.g.*, equally weighted vote). Thus, they can be significantly impacted by inaccurate configurations.

The right side of Fig. 9 shows the PR curves of random forests, two combination methods, and the top 3 highest-AUCPR basic



(a) KPI: PV. The basic detector ranking first in AUCPR is TSD MAD (win = 5 weeks), the 2nd one is historical MAD (win = 3 weeks), and the third one is TSD (win = 5 weeks).

(b) KPI: #SR. The basic detector ranking first in AUCPR is simple threshold, the 2nd one is SVD (row = 50, column = 3), and the third one is wavelet (win=3, freq=low).

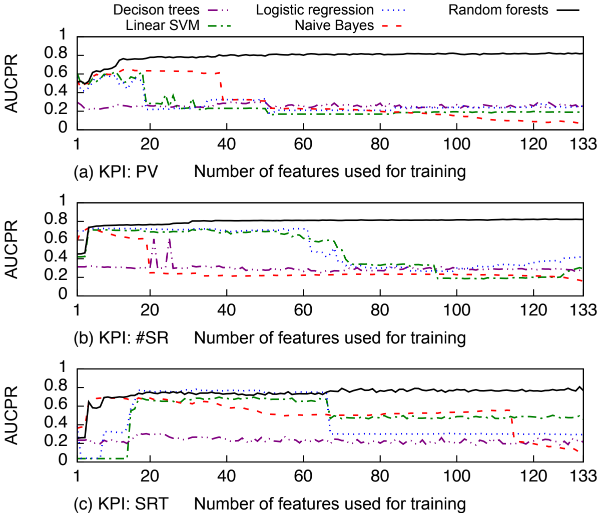
(c) KPI: SRT. The basic detector ranking first in AUCPR is SVD (row = 20, column = 7), the 2nd one is TSD MAD (win = 3 weeks), and the third one is TSD (win = 2 weeks).

Figure 9: The left side is the AUCPR rankings of different detection approaches. The right side is the PR curves. Only the top 3 highest AUCPR ranking configurations from instinct detectors are shown in PR curves.

detectors. We observe that the best basic detectors are different for each KPI, which indicates that the operators are interested in different kinds of anomalies for each KPI. Table 4 shows the maximum precision of these approaches when their recall satisfies the operators' preference (recall ≥ 0.66). We find that for all the KPIs, random forests achieve a high precision (greater than 0.8). The result shows that random forests significantly outperforms the two static combination methods, and perform similarly to or even better than the most accurate basic detector for each KPI.

5.3.2 Random Forests vs. Other Algorithms

We also compare random forests with several other machine learning algorithms: decision trees, logistic regression, linear support vector machines (SVMs), and naive Bayes. All these algorithms are trained and tested on I1 in Table 2. To illustrate the impact of irrelevant features (*e.g.*, the configurations with low AUCPR in Fig. 9) and redundant features (*e.g.*, a detector with similar parameter settings), we train these learning algorithms by using one feature for the first time, and adding one more feature each time. The features are added in the order of their mutual information [51], a common metric of feature selection. In Fig. 10, we observe that, while the AUCPR of other learning algorithms is unstable and decreased as more features are used, the AUCPR of random forests is still high even when all the 133 features are used. The result demonstrates that random forests are quite robust to irrelevant and redundant features in practice.



(a) KPI: PV

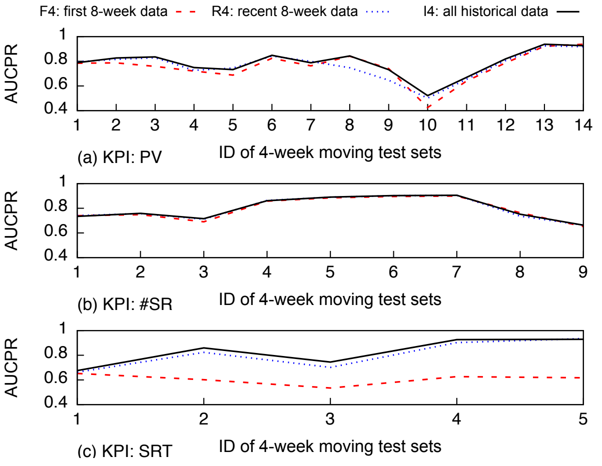
(b) KPI: #SR

(c) KPI: SRT

Figure 10: AUCPR of different machine learning algorithms as more features are used.

5.4 Incremental Retraining

After demonstrating the accuracy and stability of random forests, we want to show the effects of different training sets. We will focus only on random forests in this and the following evaluation steps. We compare three methods of generating training sets: I4, F4, and R4 in Table 2. Fig. 11 shows the AUCPR of random forests on different training sets. We see that I4 (also called incremental retraining) outperforms the other two training sets in most cases. This result is consistent with the challenge mentioned earlier that an arbitrary data set is unlikely to contain enough kinds of anomalies. In Fig. 11(b) we see that the three training sets result in similar AUCPR. This implies that the anomaly types of #SR are relatively simple and do not change much, so that they can be captured well by any of these training sets. Overall, since labeling anomalies does not cost much time (§5.7), we believe that incremental retraining is a more general and accurate method to generate training sets.

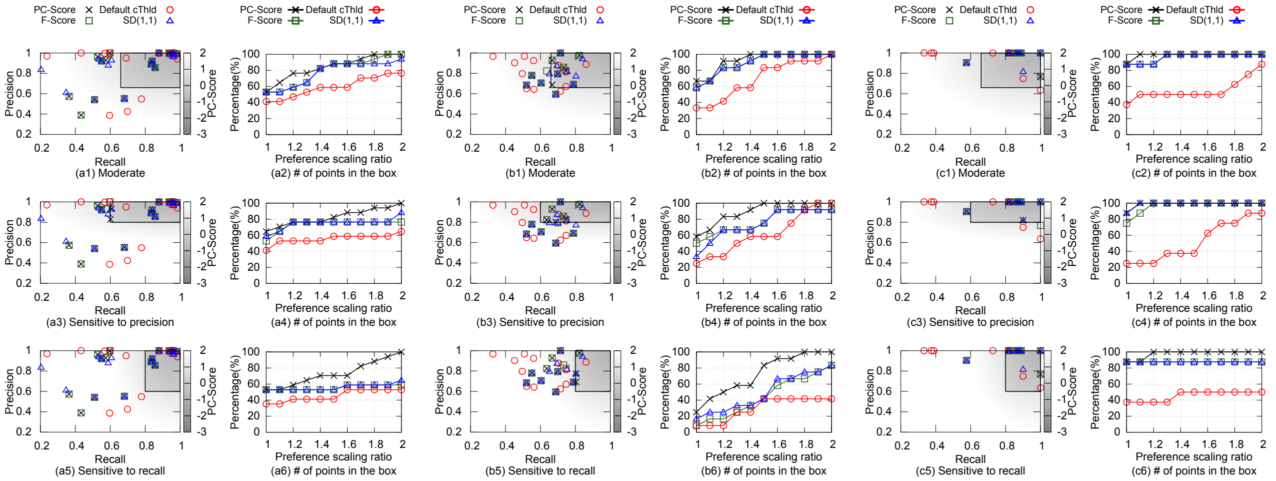


(a) KPI: PV

(b) KPI: #SR

(c) KPI: SRT

Figure 11: AUCPR of different training sets.



(a1) Moderate (a2) # of points in the box (b1) Moderate (b2) # of points in the box (c1) Moderate (c2) # of points in the box

(a3) Sensitive to precision (a4) # of points in the box (b3) Sensitive to precision (b4) # of points in the box (c3) Sensitive to precision (c4) # of points in the box

(a5) Sensitive to recall (a6) # of points in the box (b5) Sensitive to recall (b6) # of points in the box (c5) Sensitive to recall (c6) # of points in the box

(a) KPI: PV (b) KPI: #SR (c) KPI: SRT

Figure 12: Offline-fashion evaluation for the four accuracy metrics. In each sub-figure, the left side heat maps show the points of (recall,precision) produced by the four accuracy metrics, respectively. Different rows show different kinds of operators' preferences, represented by the boxes at the top-right corner. The right side line charts depict the percentage of points inside the boxes as the boxes scale up (starting from the original preferences, *i.e.*, a scale ratio of 1).

5.5 PC-Score vs. Other Accuracy Metrics

So far, we have showed the offline AUCPR of random forests without a specific cThld. Next, we evaluate different accuracy metrics that are used to configure cThlds. We compare the PC-Score we proposed against the default cThld, the F-Score, and SD(1,1). Specifically, we train and test random forests on I1 in Table 2, and let those four metrics determine the cThlds for each one-week test set. Then, we measure their performance using recall and precision of each week. Notice that, this evaluation considers an offline or oracle setting where we assume we have the test set when configuring the cThld. We will show the cThld prediction for future test set (online detection) in §5.6.

Fig. 12 shows the results of the four metrics for three KPIs, respectively. In the left-side heat maps, each point represents the recall and the precision of one week. The first row shows the results under the preference (recall≥0.66 and precision≥0.66), called a *moderate* preference. We also evaluate another two preferences: *sensitive-to-precision* (recall≥0.6 and precision≥0.8) in the second row and *sensitive-to-recall* (recall≥0.8 and precision≥0.6) in the third row. The preferences are represented by the top-right boxes in the heat maps. The right-side line charts show the percentage of the points inside the boxes (satisfying the preferences) if we use the original or lowered preferences, *e.g.*, scaling the boxes up.

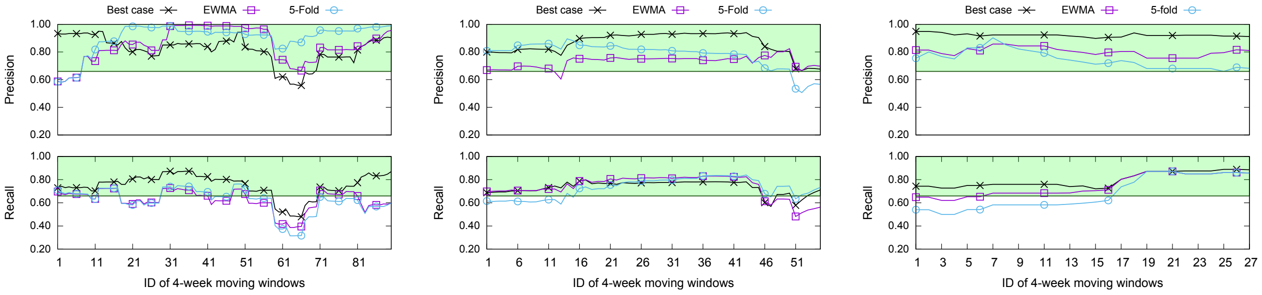
Focusing on the heat maps, we see that while the points obtained by the other three metrics remain unchanged for different preferences, the PC-Score has the ability of adjusting the recall and precision according to different preferences. Because of this advantage, we see in the line charts that PC-Score always achieve the most points inside the boxes for both the original preference and the scaled-up ones.

We also notice that it is not easy to satisfy the preference for all the weeks, because anomalies are very rare in some weeks. For example, we find that for the weeks where no point satisfies the moderate preference, anomalies are 73% and 78% fewer than other weeks for PV and #SR, respectively. Thus it is inevitable to generate more false positives in order to identify those few anomalies, leading to low precision. In addition, missing just a few anomalies can lead to an obvious degradation in recall. Fortunately, as the anomalies are few in those weeks, relatively low precision or recall would not cause a large *number* of false positives or false negatives. For example, if there are ten anomalous data points and we identify four of them (40% recall), we would miss only six anomalous data points; in such case, if we have 40% precision, we would just identify six false positives. The operators we work with suggest that they are actually OK with this small number of false positives or false negatives.

5.6 EWMA vs. 5-Fold for cThld Prediction

Previously, we show the offline evaluation of different metrics for cThld configuration, and find that PC-Score outperforms the other three. Now we evaluate the performance of online cThld prediction based on the PC-Score, which is also the detection accuracy of Opprentice as a whole. We compare the EWMA based method used by Opprentice with 5-fold cross-validation. The evaluation is similar to §5.5 except that the cThld prediction only uses the historical data (the training set) rather than the future data (the test set). As aforementioned, the result of each week can vary greatly because anomalies are quite rare in some weeks. To obtain a stable result, we calculate the average recall and precision of a 4-week moving window. The window moves one day for each step so that we can obtain a more fine-grained result. Here, we show the result under the operators' actual preference (recall≥0.66 and precision≥0.66). The recall and precision of each window are shown in Fig. 13. The offline result (called the best case here) is also shown as a baseline.

The result shows that, for PV, #SR, and SRT, the EWMA achieves 40%, 23%, and 110% more points inside the shaded regions, respectively, when compared with the 5-fold cross validation. In total, 8403 (7.3%), 2544 (2.1%), and 86 (6.4%) data points are identified as anomalies by Opprentice in the test sets (after the 9th week) for the three KPIs, respectively (not shown). We notice that there are some points falling outside of the shaded regions, such as the window between 58 and 70 in Fig. 13(a). It is mainly because the anomalies during those weeks are quite rare.



(a) KPI: PV (b) KPI: #SR (c) KPI: SRT

Figure 13: Online detection accuracy of Opprentice as a whole. The best case is obtained from offline mode. The shaded regions represent to the operators' accuracy preference (recall ≥ 0.66 and precision ≥0.66).

For example, there are only on average 4% anomalous points in the ground truth of PV data for the window between 58 and 70. However, as mentioned earlier in §5.5, in such case, a little low precision or recall would not generate many false positives or false negatives for operators. In summary, Opprentice can automatically satisfy or approximate the operators' accuracy preference.

5.7 Labeling Time vs. Tuning Time

Next, we show the time cost of labeling, the only manual job required for operators when they use Opprentice. Fig. 14 shows the operators' labeling time when they label the three types of KPI data using our tool (§4.2). The result shows that the labeling time of one-month data basically increases as the number of anomalous windows in that month. An anomalous window refers to a window of continuous anomalies derived from one label action. Among the three KPIs, SRT requires less labeling time for each month of data because it has less data points in a month as its data interval is 60 minutes. Overall, the labeling time of one-month data is less than 6 minutes. The total labeling time for PV, #SR, and SRT is 16, 17, and 6 minutes, respectively. One intuitive reason for the low labeling overhead is that the operators each time label a window of anomalies rather than labeling individual anomalous data points one by one. The anomalous windows can be much fewer (Fig. 14) than the anomalous points in the data.

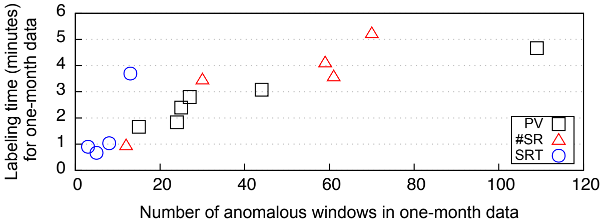


Figure 14: Operators' labeling time vs. the number of anomalous windows for every month of data.

To show the value of how Opprentice help reduce operators' manual efforts, we also present some anecdotal examples of operators' tuning time of detectors, including the time they learn the detectors and understand their parameters. We interviewed three operators from the search engine, who have experienced tuning detectors before. The first operator uses SVD, and he said it took him about 8 days to tune the detector; the second operator uses Holt-winters and historical average, and he spent about 12 days tuning these two detectors; the third operator applies time series decomposition, and he said that after the detector was implemented, it further cost him about 10 days to test and tune the detector. In the above cases, after days of tuning, only the first operator' detector works relatively well; yet, the other two operators are still not satisfied with the accuracy of their detectors, and finally abandon them. We have compared the accuracy of Opprentice with these basic detectors in §5.3.1.

Although the time reported above is not the exactly time used for tuning, it provides a basic idea of the overhead and the difficulty of manually tuning detectors. The operators we interviewed confirmed that detector tuning is very time-consuming, and they are neither interested nor feel comfortable in doing so. In summary, Opprentice can help replace the time-consuming and boring detector tuning with fast and convenient anomaly labeling.

5.8 Detection Lag and Training Time

The detection lag, in principle, consists of the feature extraction time and the classification time. We run Opprentice on the Dell PowerEdge R420 server with the Intel Xeon E5-2420 CPU and 24GB memory. The total time of extracting 133 features is on average 0.15 second for each data point. The classification takes trivial time, which is on average less than 0.0001 second per data point. Besides, the offline training time is less than 5 minutes each round. Since all the detectors can run in parallel, and training of random forests is also able to be parallelized, we believe that one can get a better performance by taking advantage of multi-threaded techniques and distributed computing on server clusters.

6. DISCUSSION

Anomaly detection is complex in practice. In this section, we discuss some issues regarding anomaly detection and clarify the scope of Opprentice.

Anomaly detection, not troubleshooting. Sometimes, although the operators admit the anomalies in the KPI curve, they tend to ignore them as they know that the anomalies are caused by some normal activities as expected, such as service upgrades and predictable social events. However, since our study focuses on identifying abnormal behaviors of the KPI data (called anomaly detection in this paper), we ask the operators to label anomalies based on the data curve itself without considering the reasons behind. Anomaly detection is a first important step of monitoring service performance. We believe that the detection results should be reported to operators and let operators decide how to deal with them, or more ideally, input into a troubleshooting system for analyzing the root causes and generating more actionable suggestions. For example, the troubleshooting system may find that the anomalies are due to normal system upgrades and suggest operators to ignore them. However, troubleshooting anomalies itself is outside our research scope.

Anomaly duration. The duration of continuous anomalies could be another important consideration of raising alarms. In this paper, we do deliberately omit this factor. One reason is that it will make our model too complex to show the core idea. Another one is that it is relative easy to implement a duration filter based upon the point-level anomalies we detected. For example, if operators are only interested in continuous anomalies that last for more than 5 minutes, one can solve it through a simple threshold filter.

Detection across the same types of KPIs. Some KPIs are of the same type and operators often care about similar types of anomalies for them [5]. For example, the PV originated from different ISPs. When applying Opprentice to such case, operators only have to label one or just a few KPIs. Then the classifier trained upon those labeled data can be used to detect across the same type of KPIs. Note that, in order to reuse the classifier for the data of different scales, the anomaly features extracted by basic detectors should be normalized. We plan to explore this direction in future work.

Dirty data. A well known problem is that detectors are often affected by "dirty data". Dirty data refer to anomalies or missing points in data, and they can contaminate detectors and cause errors of detectors. We address this problem in three ways. (a) Some of our detectors, *e.g.*, weighted MA and SVD, can generate anomaly features only using recent data. Thus, they can quickly get rid of the contamination of dirty data. (b) We take advantage of MAD [3, 15] to make some detectors, such as TSD, more robust to dirty data; (c) Since Opprentice uses many detectors simultaneously, even if a few detectors are contaminated, Opprentice could still automatically select and work with the remaining detectors.

Learning limitations. A supervised learning based approach requires labeled data for initialization. This is an additional overhead when compared with applying basic detectors directly. Fortunately, the KPI data, nowadays, are easy to obtain [1, 4, 9, 12, 14, 17, 26]. Meanwhile, labeling can also be effective and cost less time as we demonstrated earlier with out labeling tool. Another issue is that a learning based approach is limited by the anomalies within a training set. For example, anomalies can be rare, and new types of anomalies might appear in the future [16]. We solve this problem by incrementally retraining the classifier to gather more anomaly cases and learn emerging types of anomalies.

Detection accuracy. Because of the many practical challenges mentioned above, anomaly detection is a complex and challenging task. It is intractable to achieve high precision and recall all the time. We also cannot guarantee Opprentice to be able to always satisfy the operators' accuracy preference. But our evaluation shows that the accuracy of Opprentice is still promising, especially for the operators' preference in the studied service.

7. RELATED WORK

Many efforts have been put into the field of anomaly detection. Researchers have developed numerous detectors using different techniques [1–24]. In addition, researchers try to address several challenges of applying detectors in practice. (a) For auto-tuning the internal parameters of detectors, Krishnamurthy *et al.* [11] proposes a multi-pass grid search to find appropriate parameters from data. Himura *et al.* [23] searches for the parameters to maximize the ratio of detected events. In comparison, beyond detector internal parameters, we also consider automatically selecting detectors and their thresholds. (b) Some work uses ROC curves to evaluate the performance of different detectors regardless of their thresholds [9, 14, 26]. This technique is also used in our work. (c) MAD is used to improve the robustness of detectors to dirty data [3, 15]. We also implement two detectors with MAD. (d) Some solutions attempt to statically combine different detectors together [8, 21]. We compared Opprentice with them. (e) Machine learning has also been applied in anomaly detection [16, 20, 32], but it serves as basic detectors. On the contrary, we use machine learning to combine different existing detectors.

Another important challenge of anomaly detection is to obtain the ground truth to evaluate detectors. Three commonly-used solutions are: (a) using the real anomalies identified or confirmed by domain operators [1, 4, 9, 12, 14, 17, 26]; (b) generating synthetic anomalies by injecting real or pre-defined anomalies into the background data [9, 14, 18, 19]; (c) pair-wise comparisons, which treat the anomalies reported by other detectors as the ground truth [1, 8, 10, 17, 18]. Because our fundamental goal is to satisfy operators' demands, we believe that solution (a) makes more sense in this paper.

8. CONCLUSION

Applying anomaly detection to an Internet-based service has been challenging in practice. This is because the anomalies are difficult to quantitatively define, and existing detectors have parameters and thresholds to tune before they can be deployed. Our proposed framework, Opprentice, tackles the above challenges through a novel machine learning based approach. The unclear anomaly concepts are captured by machine learning from real data and operators' labels, while numerous existing detectors can be automatically combined by machine learning to train a classifier to identify the anomalies. Our evaluation on real-world search KPIs show that Opprentice consistently performs similarly or even better than the best performing basic detectors which can change for different data sets.

To the best of our knowledge, Opprentice is the first detection framework to apply machine learning to acquiring practical anomaly concepts and automatically combining and tuning diverse known detectors to satisfy operators' accuracy preference. Emerging detectors, instead of going through time-consuming and often frustrating parameter tuning, can be easily plugged into Opprentice, making the former be deployed more easily and the latter more accurate. In addition, although machine learning is a promising solution, applying it in designing a practical system needs to be careful as it presents several interesting challenges, such as imbalance class and irrelevant and redundant features. We explore and solve some of them in this paper. We believe that Opprentice provides a new schema to bridge the gap between practical operational demands and the state-of-art anomaly detectors. In our future work, we plan to work on the several issues mentioned in §6, and apply the idea of Opprentice to other network operation tasks such as automating intrusion detection and network troubleshooting.

Acknowledgement

We thank our shepherd, Renata Teixeira and the anonymous reviewers for their thorough comments and valuable feedback. We also thank Jun Zhu for his knowledge of machine learning, and Kaixin Sui for her suggestion on the detection framework. At last, we thank Juexing Liao, Guo Chen, and Xiaohui Nie for proofreading this paper.

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61472214, the National Key Basic Research Program of China (973 program) under Grant No. 2013CB329105, the State Key Program of National Science of China under Grant No. 61233007, the National Natural Science Foundation of China (NSFC) under Grant No.

61472210, the National High Technology Development Program of China (863 program) under Grant No. 2013AA013302.

9. REFERENCES

[1] Yingying Chen, Ratul Mahajan, Baskar Sridharan, and Zhi-Li Zhang. A provider-side view of web search response time. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 243–254. ACM, 2013.

[2] Ajay Anil Mahimkar, Han Hee Song, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Joanne Emmons. Detecting the performance impact of upgrades in large operational networks. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 303–314. ACM, 2010.

[3] Ajay Mahimkar, Zihui Ge, Jia Wang, Jennifer Yates, Yin Zhang, Joanne Emmons, Brian Huntley, and Mark Stockert. Rapid detection of maintenance induced changes in service performance. In *Co-NEXT*, page 13. ACM, 2011.

[4] David R. Choffnes, Fabián E. Bustamante, and Zihui Ge. Crowdsourcing service-level network event monitoring. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 387–398. ACM, 2010.

[5] Suk-Bok Lee, Dan Pei, M Hajiaghayi, Ioannis Pefkianakis, Songwu Lu, He Yan, Zihui Ge, Jennifer Yates, and Mario Kosseifi. Threshold compression for 3g scalable monitoring. In *INFOCOM, 2012 Proceedings IEEE*, pages 1350–1358. IEEE, 2012.

[6] He Yan, Ashley Flavel, Zihui Ge, Alexandre Gerber, Daniel Massey, Christos Papadopoulos, Hiren Shah, and Jennifer Yates. Argus: End-to-end service anomaly detection and localization from an isp's point of view. In *INFOCOM, 2012 Proceedings IEEE*, pages 2756–2760. IEEE, 2012.

[7] Ajay Mahimkar, Zihui Ge, Jia Wang, Jennifer Yates, Yin Zhang, Joanne Emmons, Brian Huntley, and Mark Stockert. Rapid detection of maintenance induced changes in service performance. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, pages 13:1–13:12, New York, NY, USA, 2011. ACM.

[8] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 8:1–8:12. ACM, 2010.

[9] Augustin Soule, Kavé Salamatian, and Nina Taft. Combining filtering and statistical methods for anomaly detection. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 31–31. USENIX Association, 2005.

[10] Yin Zhang, Zihui Ge, Albert Greenberg, and Matthew Roughan. Network anomography. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 30–30, Berkeley, CA, USA, 2005. USENIX Association.

[11] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247. ACM, 2003.

[12] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 71–82. ACM, 2002.

[13] Jake D Brutlag. Aberrant behavior detection in time series for network monitoring. In *LISA*, pages 139–146, 2000.

[14] Fernando Silveira, Christophe Diot, Nina Taft, and Ramesh Govindan. Astute: Detecting a different class of traffic anomalies. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 267–278. ACM, 2010.

[15] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. Antidote: Understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 1–14, New York, NY, USA, 2009. ACM.

[16] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

[17] Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 147–152, New York, NY, USA, 2006. ACM.

[18] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 219–230. ACM, 2004.

[19] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 217–228. ACM, 2005.

[20] Makoto Yamada, Akisato Kimura, Futoshi Naya, and Hiroshi Sawada. Change-point detection with feature selection in high-dimensional time-series data. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 1827–1833. AAAI Press, 2013.

[21] Shashank Shanbhag and Tilman Wolf. Accurate anomaly detection through parallelism. *Network, IEEE*, 23(1):22–28, 2009.

[22] Ayesha Binte Ashfaq, Mobin Javed, Syed Ali Khayam, and Hayder Radha. An information-theoretic combining method for multi-classifier anomaly detection systems. In *Communications (ICC), 2010 IEEE international conference on*, pages 1–5. IEEE, 2010.

[23] Y. Himura, K. Fukuda, K. Cho, and H. Esaki. An automatic and dynamic parameter tuning of a statistics-based anomaly detection algorithm. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–6, June 2009.

[24] Amazon cloudwatch alarm. http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/ConsoleAlarms.html.

[25] He Yan, Lee Breslau, Zihui Ge, Dan Massey, Dan Pei, and Jennifer Yates. G-rca: A generic root cause analysis platform for service quality management in large ip networks. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 5:1–5:12, New York, NY, USA, 2010. ACM.

[26] Daniela Brauckhoff, Xenofontas Dimitropoulos, Arno Wagner, and Kavè Salamatian. Anomaly extraction in backbone networks using association rules. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, pages 28–34. ACM, 2009.

[27] Haakon Ringberg, Augustin Soule, and Jennifer Rexford. Webclass: Adding rigor to manual labeling of traffic anomalies. *SIGCOMM Comput. Commun. Rev.*, 38(1):35–38, January 2008.

[28] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[29] Ioannis Arapakis, Xiao Bai, and B. Barla Cambazoglu. Impact of response latency on user behavior in web search. In *Proceedings of the 37th International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, SIGIR '14, pages 103–112, New York, NY, USA, 2014. ACM.

[30] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 2004.

[31] Haibo He, Edwardo Garcia, et al. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

[32] Taeshik Shon and Jongsub Moon. A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18):3799 – 3821, 2007.

[33] F. Silveira and C. Diot. Urca: Pulling out anomalies by their root causes. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.

[34] Øivind Due Trier, Anil K Jain, and Torfinn Taxt. Feature extraction methods for character recognition-a survey. *Pattern recognition*, 29(4):641–662, 1996.

[35] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. John Wiley & Sons, 2013.

[36] Auto arima. http://www.inside-r.org/packages/cran/forecast/docs/auto.arima.

[37] Ramón Díaz-Uriarte and Sara Alvarez De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):3, 2006.

[38] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[39] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.

[40] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1ÂÍC2):273 – 324, 1997. Relevance.

[41] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[42] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 339–350. ACM, 2013.

[43] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, and He Yan. Modeling web quality-of-experience on cellular networks. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 213–224, New York, NY, USA, 2014. ACM.

[44] Yoav Freund, Robert Schapire, and N Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[45] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[46] Neil J Perkins and Enrique F Schisterman. The inconsistency of "optimal" cutpoints obtained using two criteria based on the receiver operating characteristic curve. *American journal of epidemiology*, 163(7):670–675, 2006.

[47] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14, pages 1137–1145, 1995.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Adam Lazur. Building a billion user load balancer. In *Velocity Web Perfomance and Operations Conference*. O'REILLY, 2013.

[50] Kendrick Boyd, Kevin H Eng, and C David Page. Area under the precision-recall curve: Point estimates and confidence intervals. In *Machine learning and knowledge discovery in databases*, pages 451–466. Springer, 2013.

[51] Hanchuan Peng, Fuhui Long, and Chris H Q Ding. Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy. *IEEE Trans. Pattern Anal. Mach. Intell. ()*, 27(8):1226–1238, 2005.