AESOP: Automatic Policy Learning for Predicting and Mitigating Network Service Impairments

Supratim Deb, Zihui Ge, Sastry Isukapalli, Sarat Puthenpura,
Shobha Venkataraman, He Yan, Jennifer Yates
AT&T Labs, Bedminister, NJ
[supratim,gezihui,sastry,sarat,shvenk,yanhe,jyates]@research.att.com

ABSTRACT

Efficient management and control of modern and next-gen networks is of paramount importance as networks have to maintain highly reliable service quality whilst supporting rapid growth in traffic demand and new application services. Rapid mitigation of network service degradations is a key factor in delivering high service quality. Automation is vital to achieving rapid mitigation of issues, particularly at the network edge where the scale and diversity is the greatest. This automation involves the rapid detection, localization and (where possible) repair of service-impacting faults and performance impairments. However, the most significant challenge here is knowing what events to detect, how to correlate events to localize an issue and what mitigation actions should be performed in response to the identified issues. These are defined as policies to systems such as ECOMP [1].

In this paper, we present AESOP, a data-driven intelligent system to facilitate automatic learning of policies and rules for triggering remedial actions in networks. AESOP combines best operational practices (domain knowledge) with a variety of measurement data to learn and validate operational policies to mitigate service issues in networks. AESOP's design addresses the following key challenges: *(i)* learning from high-dimensional noisy data, *(ii)* capturing multiple fault models, *(iii)* modeling the high service-cost of false positives, and *(iv)* accounting for the evolving network infrastructure. We present the design of our system and show results from our ongoing experiments to show the effectiveness of our policy leaning framework.

CCS CONCEPTS

•**Computing methodologies** → **Rule learning;** *Supervised learning by classification;* •**Networks** → *Network management;*

KEYWORDS

Policy Learning; Network Management; Supervised Learning

1 INTRODUCTION

Today's networks serve hundreds of petabytes of traffic every day. As consumer demand for high bandwidth services such as video continues to explode, network traffic volumes are expected to continue to grow at a rapid pace. Supporting such a continued rapid growth in network traffic volumes without a corresponding increase in network operations staff requires increasingly efficient management and control of increasingly complex networks.

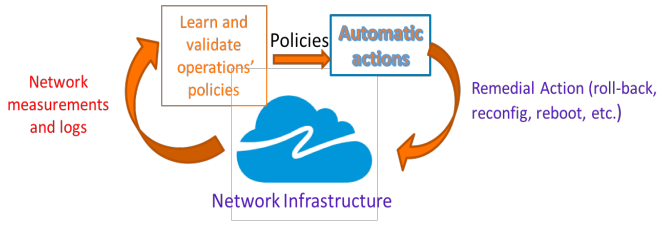


Figure 1: Policy learning based automation loop for network service mitigation

Fault and performance management constitutes a significant portion of the day-to-day operations activities in large networks or distributed systems [8]. As the name suggests, fault and performance management deals with the fault or performance degradation conditions occurring in networks. The operational steps include timely detection of the fault or performance degradation conditions in the system, localizing the faulty or underperforming network elements, and devising and deploying mitigation measures in the system – e.g., by performing a reboot on a network device, or switching traffic to a secondary path.

Traditional network management has relied on considerable domain expertise and operational knowledge accumulated in Operations personnel through years of experience. To improve customer service experience and reduce operational cost, network service providers are heavily investing in advanced automation technologies to facilitate the real-time response to fault and performance management. For example, in 2016, AT&T unveiled ECOMP [1], a cloud-based scalable software platform for *policy driven automation* of network management functions. These automation systems would be governed by a set of operations' policies which in the case of fault and performance management define the trigger conditions and the procedures to automatically respond to service impairments. Traditionally, the policies or rules triggering the responses to service impairments are defined through domain expertise. However, this is typically incomplete and hard to capture as this knowledge is distributed across large number of operation personnel. Instead, we propose using a machine-learning based approach to identifying these policies. The input to our learning based system comes from vast numbers of logs capturing the actions that network operators took in the network along with high-dimensional measurement

KDD'17, August 13–17, 2017, Halifax, NS, Canada.

DOI: http://dx.doi.org/10.1145/3097983.3098157

time-series data capturing the conditions that existed in the network as these actions were performed. Our goal in this work is to design a system that automatically learns and validates policies for the automated detection, localization and repair of fault and performance impairments in networks. We term this problem as *policy learning*. The complete automation cycle enabled by our policy learning approach is illustrated in Figure 1. For the remainder of this paper, we use cellular networks as a use case to illustrate our design and for evaluation purposes. However, our framework and core design principles of policy driven automation are applicable more generally to other types of networks and distributed systems - comprised of physical and/or virtual network functions (network elements).

In this work, we focus on policies for mitigating network service impairments. Such policies would specify a remedial action, such as rebooting a cellular base station, resetting an antenna element, etc - actions which are taken in response to identified impairments. The typically vast network scale results in the same underlying network impairments typically appearing *repeatedly* - albeit in different locations and at various times of day. Impairments caused by a common software bug or underlying root cause would be repaired via a common remedial action. Given the complexity of the software and network conditions, such responses to given network conditions are often determined by operators through experience, with this knowledge (inconsistently) distributed across possibly large numbers of operators. If there is a sufficient number of these impairment instances, we hypothesize that we could automatically learn the policies that define the conditions under which different remedial actions should be invoked. With these learned policies, the remedial actions could be fully automated[1].

1.1 Challenges

In practice, however, learning operational policies for network management and control is not a pure data mining exercise. A system that learns policies to mitigate network service issues needs to have a modular and flexible architecture fusing operational knowledge into the learning pipeline to address the following challenges:

(1) **Taking incorrect remedial actions:** A huge challenge to policy learning comes from the typically high service cost of an incorrect action. A remedial action that is just unnecessary will still likely impact customer experience during the mitigation period; an incorrect action may result in additional complications. Thus the false positives have to be very low so that the service downtime is acceptable. Our goal instead is to learn policies that can automate as many correct remedial actions as possible while keeping the false positives extremely low.
(2) **Multiple faults:** A second challenge comes from multiple underlying types of impairments (fault or performance degradation) that may require similar remedial actions. The available remedial action log does not have annotation of the underlying fault or service degradation that was the reason that the action was taken. Thus, we would like to learn a collection of policies where each policy implicitly captures an underlying type of network impairment and its associated remedial action.
(3) **Evolving heterogeneous infrastructure:** The network infrastructure is constantly in a state of flux, with software upgrades, configuration changes, new technologies, new types of devices and new applications appearing on an ongoing basis. Any policy for a remedial action is typically specific to an operational environment (e.g., a combination of software version, configuration, services activated, protocols and devices supported, etc.) under which it applies. Thus, we would like to learn policies for different operational environments and diverse implicit impairment conditions.

In addition, the policies learned should assist the experts in finding permanent fixes for the faults. Domain experts typically do this through root-cause analysis. The learned policies can aid experts significantly by identifying relevant measurement counters for further investigation. Further, tracking the operational environment can also help experts with root cause analysis, by detecting broad trends in automated remedial actions. For example, if a number of automatically detected service issues relate to network elements with a specific software version and a specific type of service degradation, the vendor could be contacted to identify a permanent fix.

1.2 Contributions

In this paper, we describe AESOP (*Automatic Extraction of Signatures for Operation Policies*), a system for automatically learning policies for mitigating network service impairments. Our key insight is the following: while raw measurement data and logs do not reveal the underlying faults, we can design higher-level statistically-derived *symptoms* that can be used as pivots to learn policies for different operational environments. To the best of our knowledge, we present the first platform for automated policy learning in networks. In particular, our main contributions are as follows:

Policy learning framework: We present a novel data-driven framework and architecture for automatically learning policies for mitigating network service impairments. Our system is modular and flexible enough to configure based on different use-cases.

Modeling and algorithms: We develop a data-driven system model that reflects key principles of network operations, models service degradation in a systematic manner, and learns operations' policies for different operational environments and different types of service degradation.

Experimental results: We present results from our ongoing experiments to illustrate the effectiveness of our system in the context of cellular networks. We show that the policies learned can correctly predict a sizable percentage (up to around 50%) of remedial actions from a target set of predictable instances.

2 DATA DRIVEN POLICY LEARNING

Our goal is to design a system that can learn policies for mitigating network service issues. Our core design principles and framework apply to all kinds of networks including SDN, NFV based networks, wireline and cellular networks. However, we will illustrate our design with the aid of a cellular LTE network setting. To this end, we first introduce some terminology for cellular LTE networks.

[1]We focus only on the learning aspects in this work. The system for automating the remedial action based on a learned policy (e.g., a system for automatically rebooting a base station) is out of the scope of this paper.

In LTE networks, the main network element responsible for wireless transmission and reception is known as the *eNodeB* (the so-called "cell towers"). Each eNodeB has multiple transmitters; a mobile device is associated with any one of the eNodeB's transmitters. A wide range of measurements are reported by the eNodeB; these measurements are collected at the eNodeB and sent to data centers periodically.

2.1 Network Measurement Data

The different data types used to present our system design and evaluation are as follows:

(1) **Measurement counters:** These are raw performance measurement counters reported by eNodeBs. These counters are reported at fixed measurement intervals (typically 15 min), thereby creating time-series data. Example counters include the number of connection attempts in a measurement window, the number of call failures during hand-off to a neighboring cell, the number of bytes downloaded, etc. Each eNodeB reports more than a thousand measurement counters.
(2) **Service related Key Performance Indicators:** For each eNodeB, service related Key Performance Indicators (service KPIs) are measurements that show quality of service delivered to end-users. In general, there are other KPIs that are not directly related to service. However, for the purpose of this paper we are mainly interested in service KPIs and we simply refer to them as KPIs in rest of the paper. KPIs are typically based on a formula that takes as input the raw measurement counters described in the preceding paragraph. Some example service KPIs include downlink throughput, call accessibility percentage (i.e., a measure of the fraction of attempted calls that were successful), call retainability (i.e., a measure of the fraction of calls that were not dropped) etc. The important point to note is that, frequently used KPIs are much fewer in number, typically no more than a few tens of time series. For ease of exposition, we will assume that higher values of service KPIs implies better service quality. This is true for the examples of service KPIs given in this paragraph.
(3) **Remedial action logs:** These are specific network logs that record details (timestamp, steps etc.) of remedial actions taken at different eNodeBs by expert operations engineers. When an eNodeB experiences degraded performance for an extended period of time, a ticket is created to alert a network engineer, who takes a suitable remedial action, e.g., software reset or hardware reboot of the eNodeB. These actions are logged using network management software.
(4) **Network configuration:** The network configuration data captures how each element is configured, including the software version and the parameters that specify the network elementfis functionality. Since these configurations change over time, the data also contains the date associated with each configuration snapshot. We will refer to the software version and configuration template combination as the *operational environment*.

Remark. *We present our system and evaluation in the context of the preceding types of data. However, our system architecture and policy learning framework is general; it can incorporate other types of data including alarms, syslogs, service related tickets, etc.*

Use case specific meta-data: So far we have described some different types of data that are typically extensively leveraged for operating a network. However, there are additional properties of the data that depends on the specifics of a given network and service. These properties can be captured as meta-data and the design of AESOP accounts for this meta-data. The specification of this meta-data can be configured by a domain expert on a per use case basis. Precisely, our system accounts for two types of use case specific meta-data, namely *primary KPIs* and *measurement categories*. These are as follows in the context of cellular 4G LTE network:

- **Primary KPI:** These are the most important KPIs that accurately reflect the service quality of an eNodeB. Put simply, if the primary KPIs are not degraded, the eNodeB is considered to be in good health. Domain experts can hand-pick the list of primary KPIs (typically around ten or so) based on expert knowledge of critical service metrics. In a latter section, we show how the choice of primary KPIs is used to define the notion of a *symptom* associated with a potentially degraded eNodeB.
- **Measurement categories:** Although there are over a thousand performance measurement counters, many of them present different performance aspects of a common underlying physical phenomena or a common networking protocol. Thus, measurement counters can be categorized into correlated groups within which many counters will often provide redundant information. The categorization can be obtained in a number of ways: labeling by a domain expert, common keywords in counter names, originating protocol or phenomena of the counters, etc. For policy learning, categorization has two benefits: *(i)* dimensionality reduction through a two-stage process (see Section 3) that isolates important features in each category and pools them together, and *(ii)* identification of higher-level components responsible for the network impairment, thus enabling easier root-cause analysis.

Scale of data: Depending on the network, the scale of the data could vary and is typically very large. In the cellular network studied here, there are around 900 million distinct measurements per day for a large US metropolitan area (e.g., New York City).

2.2 Problem Formulation

Our goal in this work is to learn policies for network problem mitigation by combining data-driven intelligence with domain knowledge. To this end, we formalize the notion of a policy in the context of our work. Before we do that, we will introduce some key notations. We present our problem in terms of the specific application of rebooting eNodeBs. However, the framework is general and can be applied to other areas of a network, different mitigation actions and different network data. For the use case analyzed here, we are given the following historical data:

(1) Network measurement counters $\mathbf{x_c}(e,t)$ as a function of eNodeB e and time t, along with measurement KPIs $\mathbf{x}_{kpi}(e,t)$, where $(\mathbf{x}_c, \mathbf{x}_{kpi}) \in \mathbb{R}^m$.
(2) Expert aided network logs $a(e,t)$ which represent actions that were taken on eNodeB e at time t where $a(.) \in \mathbb{A}$ and $\mathbb{A}$ is a finite set of remedial actions. To keep the notation generic, whenever there is no remedial action taken on an eNodeB (during normal

working conditions), we consider this as a special action that can be termed "no-remedial-action required."

Policies for a remedial action: In simple terms, a policy for a remedial action is a mapping from network measurement counters/KPIs to a *possible* remedial action on a given network element (eNodeB in our specific use case). However, the applicability of each policy to a network element (eNodeB) depends on *(i)* the operating environment (software version and configuration) of the eNodeB, and *(ii) symptoms* or degradation condition (to be made precise in the next section) of the eNodeB if any. More formally, each policy is characterized by the following:

- *Policy domain:* This defines the specific operating environment of an eNodeB for which a policy is applicable and the specific symptoms or degradation conditions of the eNodeB. We will provide a more formal definition of symptom in a later section, but intuitively each symptom specifies the set of primary KPIs that were severely degraded thus requiring a possible remedial action.
- *Policy function:* This function is a mapping from $(\mathbf{x}_c, \mathbf{x}_{kpi})$ to $a \in \mathbb{A}$, the set of actions. Note that, WLOG, the set of actions also includes "no-remedial-action required."

Problem Statement: The policy learning problem can be stated as follows: *learn all possible policies (policy domain and function) based on historical network counters/KPIs* $\mathbf{x_c}(e,t)$ *and* $\mathbf{x}_{kpi}(e,t)$ *along with action logs* $a(e,t)$. The learning objective is to ensure that each policy function maps to an accurate remedial action that can fix the service degradation while the false positive rate of taking a remedial action (excluding "no-remedial-action" required) is less than ϵ, a small number. This is because an incorrect action on an eNodeB can lead to small but non-negligible loss of service on the eNodeB when the eNodeB is under repair.

We will show in the next sub-section how this can be cast as a classification problem after suitable data-processing steps.

2.3 AESOP Overview

As described earlier, using machine learning for this problem needs to address several challenges: high-dimensional noisy data, multiple fault models, low false positives, rapidly-changing infrastructure and more. To address these challenges, we take the following data-processing steps to construct training examples:

(1) *Computing base-line summaries:* This step computes spatio-temporal statistical summaries by computing statistics (mean, median, quantiles) for each measurement counter/KPI and for each eNodeB at different times of the day. This step is useful to subsequently convert raw measurements into a normalized anomaly score.
(2) *Data normalization via anomaly scoring:* The main functionality of this step is to perform normalization of raw measurement values by converting raw data into anomaly scores.
(3) *Symptom modeling:* This step assigns *symptom*, if any, to every measurement data instance. Since the action logs do not have annotation for possible fault, the symptoms allow us to group measurement data corresponding to different remedial actions in a systematic manner. The symptoms are assigned based on combinations of degraded primary KPIs. Subsequently, this step constructs the training examples by suitable data cleansing, downsampling, and grouping of training examples; each group will have a common set of policies.
(4) *Feature extraction and classification:* This has two machine learning functionalities. First, this module performs dimensionality reduction of the data by combining expert-aided feature categorization with standard feature extraction techniques. Second, it learns suitable classifiers so that the desirable properties in our problem statement in Section 2.2 are satisfied.

These steps learn the policies periodically (say, once in a week). In addition, there is also a separate module that validates the policies learned. Figure 2 shows the interaction of the different system modules carrying out the above steps.

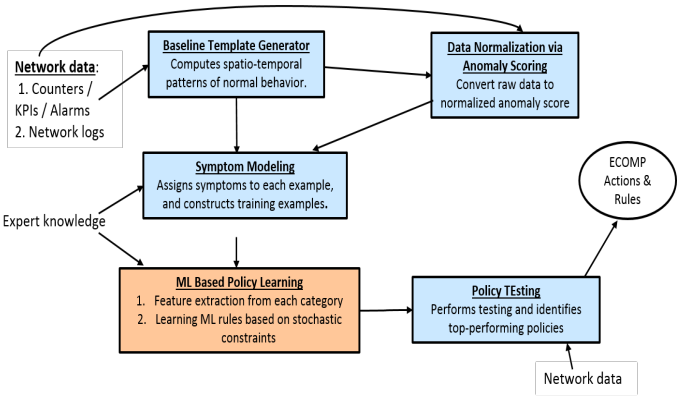


Figure 2: Key system modules of the policy learning framework

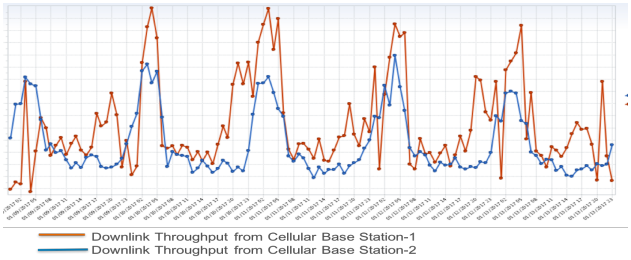


Figure 3: The behavior of a key KPI (downlink throughput) from two different cellular eNodeBs on a cellular operator's network. The exact values on the y-axis are not shown as the data is proprietary.

3 AESOP FRAMEWORK

We now describe the models and algorithms used in the AESOP framework for processing the network data and learning policies.

3.1 Computing Statistical Baselines

The statistical baseline for each counter and KPI must account for the underlying spatio-temporal pattern. Cellular network data is known to exhibit time-of-day and day-of-week patterns for many key metrics. As an illustration, Figure 3 depicts the downlink throughput for two different eNodeBs over a one week period, illustrating the clear time varying nature of the throughput. We therefore compute statistical baselines using the following steps:

(1) *Binning:* For each data sample from a measurement counter/KPI, we associate a spatio-temporal *bin* depending on three parameters: hour of day, day of week, and eNodeB of the data. The binning is done over a time-window that is typically a few months to strike a balance between capturing slow changes in network behavior and having sufficient samples in each bin for computing reliable statistical estimates. Also, prior to the binning process, missing data is given a special label.
(2) *Computing statistics:* Since we do not want to include any anomalous data (due to fault or other planned operational activities) in our statistical baselines, we first do data cleansing as follows: exclude missing data and filter out data that corresponds to a time-window (24 hours typically) within which the eNodeB underwent some remedial action or when the eNodeB was in maintenance mode. Once the data cleansing is done, the ensemble of remaining data in each bin is used to compute different statistics: mean, standard deviation, quantiles $(10, 25, 50, 75, 90)$.

3.2 Data Normalization via Anomaly Scoring

Since the data can have very different ranges across measurement counters/KPIs, originating eNodeBs and measurement times, we normalize the measurements to get all data in the same scale. Many machine learning algorithms rely on the different dimensions to be in the same scale. Normalization also helps us in feature extraction by allowing us to compare *importance weight* of different measurements in a meaningful manner. Since our goal is to learn policies for remedial actions that fix service problems, as a design choice, we perform this normalization via anomaly scoring. To motivate this for cellular networks, consider the following example: since cellular download speed depends on network load and eNodeB configuration, a download speed of 2 Mbps could be perfectly normal at 9 AM in a busy business area but highly abnormal at 4 PM on weekends in a residential area. To learn data-driven policies, we need to have a consistent anomaly score that reflects these two scenarios.

To compute the anomaly score, we make use of IQR (inter-quantile range) based robust statistics combined with notions of outliers and extreme outliers in data. For each measurement data we first retrieve the statistics of the associated spatio-temporal bin as defined in Section 3.1. Suppose, for a given measurement x, that the 25^{th} and 75^{th} quantiles for the corresponding bin are $Q1$ and $Q3$ respectively. We also have $IQR = Q3 - Q1$. We wish to find a mapping so that the following is satisfied: (i) all anomaly scores lie in the range $[0, 1]$, (ii) extreme outliers defined as $3 \times IQR$ above $Q3$ or $3 \times IQR$ below $Q1$ are mapped to the right and left end-points of the range, 1 and 0 respectively. One such mapping that satisfies the above is as follows:

$$f_{as}(x) = \begin{cases} \max\left(0, 0.5(1 - \frac{Q1-x}{3IQR})\right) & x < Q1 \\ 0.5 & x \in [Q1, Q3] \\ \min\left(1, 0.5(1 + \frac{x-Q3}{3IQR})\right) & x > Q3 \end{cases} \quad (1)$$

Indeed, $f_{as}(Q1 - 3IQR) = 0$ and $f_{as}(Q3 + 3IQR) = 1$.

Our above choice is robust to skew in data statistics (unlike a z-score based statistic) and is also computationally light (unlike sophisticated measures based on time-series anomaly detection). This is an important requirement, given the dimension of the data (over a thousand per eNodeB per measurement instance).

Capturing time-series history: We also augment the features by computing the time-average statistics of each normalized measurement. Specifically, we compute rolling statistics (median and mean) of the normalized anomaly scores over a fixed time window, set as 2 hours.

In summary, this module performs the following functions:

(1) For each dimension in the measurement vector $(\mathbf{x_c}(e,t), \mathbf{x_{kpi}}(e,t))$, the module computes the anomaly scores given by (1). Denote the vector of the anomaly scores by $(\mathbf{z_c}(e,t), \mathbf{z_{kpi}}(e,t))$.
(2) For each dimension in $(\mathbf{z_c}(e,t), \mathbf{z_{kpi}}(e,t))$, the module computes the windowed running median and mean values over a time window $[t-W, t]$ where typically $W = 2$ hrs which corresponds to the past eight measurements since measurements are sent once every 15 min. The choice of W reflects the fact that we wish to learn policies that can rapidly detect service issues. W is a configurable parameter that should be set based on expert operational insights.

Missing data: As discussed in Section 3.1, missing measurements could be indicative of a problem or could instead be the result of a data collection issue. Thus we assign a special symbol for missing data. We also tested encoding strategies like one-hot encoding. However, our experience was that this considerably increases the dimensionality (already into the thousands) without providing appreciable gains in effectiveness of our models.

3.3 Symptom Modeling

A remedial action on a network element is typically taken in response to an underlying fault that causes some service problem. However, the action log which we are using to capture the dates, times and locations of given operator actions (e.g., eNodeB reboots) does not capture any information regarding why an action was taken by a network operator. To overcome this challenge, AESOP's design makes use of the following insight: *each underlying root-cause manifests as a degraded condition on some subset of the most important KPIs. We refer to these KPIs as the primary KPIs.* The primary KPIs are specified as inputs to AESOP - meta-data provided by domain experts. Thus, a *symptom* is simply some combination of degraded primary KPIs. To assist in a formal definition of a symptom, we first define the degradation score of a KPI.

Definition 1 (KPI degradation score). *Given windowed running-median anomaly score of a measurement KPI $\hat{z}(t)$, degradation score of the KPI is defined by $Score_d(\hat{z}(t)) = \sum_{k=0}^{K} \mathbb{1}_{(\mu - \hat{z}(t) \geq L_k)}$ where $\mu = 0.5$ is the nominal anomaly score and L_k's are discretization levels (e.g, $L_0 = 0.1, L_1 = 0.2, L_2 = 0.3$).*

Definition 2 (Measurement symptom). *Given a time-windowed running median anomaly score of a primary KPI-vector $\hat{\mathbf{z}}_{\mathbf{kpi}}(t) = (\hat{z}_1(t), \hat{z}_2(t) \ldots)$, the underlying* symptom *of the measurement is characterized by two properties: (i) a symptom score given by the maximum degradation score $MaxScore_d(\hat{\mathbf{z}}_{\mathbf{kpi}}) = \max_i Score_d(\hat{z}_i)$, and (ii) symptom KPIs given by the* **set of KPIs** *with the maximum degradation score, i.e. the set $\{i : Score_d(\hat{z}_i) = MaxScore_d(\mathbf{z_{kpi}})\}$*

Intuitively, the notion of a symptom captures the directions of the KPI vector that are most abnormal (in a probabilistic order of magnitude sense). When all KPIs are within $Q1$ and $Q3$, the symptom has an associated degradation score of zero.

We now describe how we construct training examples. The training examples consist of two sets of records - measurements that were followed by a remedial action and measurements during normal network behavior. We perform the following steps:

(1) *Symptom computation:* For each measurement, we compute the associated symptom based on the preceding definition.
(2) *Extracting successful remedial actions:* Though remedial actions are taken to mitigate service impairments, sometimes a given remedial action may not fix the service problem in which case an operations engineer may try an alternative remedial action. Since we wish to learn patterns for remedial actions that fixed the service problem, it is imperative to compute the remedial actions (from the set of all remedial actions in remedial action log) that successfully mitigated service impairments. To extract the successful remedial actions, for each remedial action we compute an aggregate[2] degradation score across KPIs in fixed length time windows before and after each remedial action. Any remedial action that did not result in an improvement based on this aggregate score is removed from the training data set.
(3) *Downsampling*: When there is a service problem at an eNodeB, it gets manifested in terms of different symptoms at different times (prior to the remedial action). This is because degradations for different KPIs have some degree of correlation. This raises an important question: which instances and corresponding symptoms should be represented in the training data for a particular remedial action? Since our goal is to learn policies for mitigating service impairment, a natural choice is to use those instances when the service impairment was most severe. In our framework, we construct a single training example per remedial action as follows: we pick the measurement corresponding to the worst aggregate degradation score in the T_h=12 hr period leading up to the remedial action. For constructing training data corresponding to measurements of normal behavior, we randomly sample data from normal behavior such that there is at most one measurement from an eNodeB on a single day; this ensures statistical independence of the training examples.
(4) *Grouping for identifying policy domains:* Finally, we group the training examples from each remedial action based on the symptom and additional data related to the operational environment of an eNodeB. The key idea is to learn policies for each group individually, as service problems in each group are likely to have a common underlying pattern. Note that, a group reflects a policy domain corresponding to policies learned for the group.

Symptom pooling: Ideally we wish to learn policies for each underlying symptom. This is a challenge for symptoms without sufficient training data to learn patterns. For example, suppose for an operational environment, there are (i) 100 remedial actions corresponding to a symptom with KPI-1 and KPI-2 both simultaneously worst hit with a degradation score of 3, and (ii) there are 15 remedial actions corresponding to symptoms where KPI-1, KPI-2 and KPI-3 are worst hit with a degradation score of 3. In this case, we pool the latter training examples with the former. In general, whenever the size of training data set is less than a threshold for some symptom $s1$, we pool it with another symptom $s2$ such $s1$ and $s2$ have common degraded KPIs; this process can be iterated a few times if required.

[2] Aggregate degradation score is the normed ($L2$ norm) score across individual KPI degradation scores.

3.4 Feature Extraction and Classification

We will describe our models assuming a single remedial action. The extension to multiple remedial actions is similar to extending 2-class classification to a multiclass classification. Without loss of generality, we consider data corresponding to a *single group* of training examples from the remedial action since we wish to learn the policies for each group separately. Recall that each group (policy domain) is characterized by a symptom, software version and configuration as outlined in Section 3.3. In this section, we say measurements to mean "measurements normalized via anomaly scores."

After constructing the training examples in each group using the data-processing steps discussed in the previous section, let $z_i, i, 1, 2, \ldots, m$ be the set of measurements, where $i \in [1, m_1]$ corresponds to measurements prior to the remedial action (class-R) and $i \in [m_1 + 1, m]$ corresponds to measurements during normal network behavior (class-N). Let y_i be the binary variable denoting the two classes, i.e., $y_i = 1$ for $i \in [1, m_1]$ and $y_i = 0$ for $\in [m_1 + 1, m]$. We will also use the notation $\mathcal{D}^+ = \{z_i : i \in [1, m_1]\}, \mathcal{D}^- = \{z_i : i \in [m_1 + 1, m]\}, \mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$. Our goal is to compute a mapping $y = h(z)$ so that we maximize the true positives subject to low false positives. Denoting $\Pr_{\mathcal{D}^+}(h(z)) = \Pr(h(z) \mid z \in \mathcal{D}^+)$ and $\Pr_{\mathcal{D}^-}(h(z)) = \Pr(h(z) \mid z \in \mathcal{D}^-)$, we wish to solve the following problem:

$$\max_{h:\mathcal{D}\to\{0,1\}} \Pr_{\mathcal{D}^+}(h(z) = 1) \quad \text{s.t. } \Pr_{\mathcal{D}^-}(h(z) = 1) \leq \epsilon$$

In log-domain, the above problem is equivalent to

$$\max_{h:\mathcal{D}\to\{0,1\}} \log \Pr_{\mathcal{D}^+}(h(z) = 1) \quad \text{s.t. } \log \Pr_{\mathcal{D}^-}(h(z) = 0) \geq K_\epsilon \,, \tag{2}$$

where $K_\epsilon = \log(1 - \epsilon)$. Though this is a constrained optimization problem, this can be cast into an unconstrained problem using Lagrangian duality principles [4]. Under suitable convexity assumptions [4], (2) is equivalent to solving the following unconstrained problem:

$$\min_{\lambda\geq 0} \max_{h:\mathcal{D}\to\{0,1\}} \mathcal{L}(h(.);\lambda)$$
$$\mathcal{L}(h(.);\lambda) = \log \Pr_{\mathcal{D}^+}(h(z) = 1) + \lambda(\log \Pr_{\mathcal{D}^-}(h(z) = 0) - K_\epsilon) \,, \tag{3}$$

where $\mathcal{L}(.)$ is the Lagrangian of the original constrained problem. Note that for a given value of λ, the expression in (3) is exactly the same as log-likelihood of the training data with the class-N examples weighted by λ. Thus, we have a standard classification problem where the optimal classifier is computed based on maximizing log-likelihood of the training data; also, there is an additional hyper-parameter λ which needs to be optimized.

We now describe our feature extraction and learning steps.

3.4.1 Feature Extraction. The measurement data has around 1100 counters and around 2200 dimensions once we augment the feature space with rolling time window based statistics (median or mean). Thus, it is very important to select the right features

to learn policies that are effective. Rather than applying a feature extraction methodology to the entire pool of features, we can make use of expert aided measurement categories as each category has features that relate to a common underlying phenomena or protocol. For example, in cellular networks, a very important KPI is the fraction of calls not dropped when mobile users move across cell boundaries. When a call is dropped, it is captured by multiple measurement counters corresponding to different protocols and events. Expert knowledge provides a measurement category for all such measurements as meta-data. Thus, for symptoms related to this KPI, we can focus on this set of features and perform feature extraction much more efficiently.

To this end, our feature extraction step identifies the top features in each measurement category and then pools them together. This ensures that the most important feature each *related* group of features are used in learning. See Section 3.5 on how this approach lends to interpretable policies.

For extracting features from a category of measurements we proceed as follows. First, we project training example-i z_i into a lower dimensional space consisting of measurements belonging to this category. Next, we use stability selection [15] which repeatedly uses a random subset of examples and a random subset of features to perform randomized logistic regression with L1 regularization; the repetitions are aggregated to produce an importance weight for each feature. Once the feature weights are chosen, we can drop all features with weight below a threshold relative to the most important feature in a measurement category.

3.4.2 Classification and regularization. The classification objective is to compute whether a given measurement instance can be classified as normal behavior or as a service issue fixable through the remedial action. We will use logistic-regression based classifier for our purpose as follows. According to (3), this boils down to maximizing

$$Obj(\mathbf{w} \mid \lambda, \beta) = \log \Pr_{\mathcal{D}^+}(h(z) = 1) + \lambda \log \Pr_{\mathcal{D}^-}(h(z) = 0) - \Omega(\mathbf{w})$$

where $y = h(z)$ is modeled by a sigmoid logistic function as

$$\Pr(h(z) = 1) = \frac{1}{1 + \exp(-w_0 - \mathbf{w}^T \mathbf{z})} \ . \tag{4}$$

There are two hyper-parameters of the classification step: λ which is the weight of class-N in (3), and a cut-of factor for feature selection β which decides relative importance of the most and the least important feature from each measurement category. The term $\Omega(\mathbf{w})$ is a regularization term for which we choose elastic net[3]. Elastic net provides robustness by striking a balance between sparsity of features and capturing groups of correlated features.

Tuning classifier for low false positives: Once the model is trained by computing the optimal $\mathbf{w}$, we need a cut off for class probabilities. We choose this to satisfy the false positive constraint. Denoting $p(h(z_i))$ as the value of $\Pr(h(z_i) = 1)$ for the computed optimal $\mathbf{w}$, this can be done as follows: (i) based on the trained model, sort the values of $p(h(z_i))$ for all i belonging to class-N, (ii) compute the $(1-\epsilon)^{\text{th}}$ quantile of the sorted values of $p(h(z_i))$ from class-N.

[3]Elastic net is a linear combination of $L1$ and $L2$ regularization.

Hyper-parameter search: The previous steps are for a given choice of hyper-parameter. The suitable value of hyper-parameter combination (λ, β) is chosen through a grid-search over reasonable values. Each choice of the tuple is evaluated through a K-fold cross-validation where $K = 10$.

3.4.3 Boosting. Training data corresponding to a given symptom within a group could have multiple underlying patterns leading to the same remedial action. Thus, we would like to compute multiple policies within each group. A natural way to express this would be to use a disjunction (i.e., logical *OR*) of multiple policies within each group. Indeed, even in manual operations, upon dissecting the data, the next step is to check if one of a few simplistic conditions are violated. We use a boosting-like [7] approach to extend our classification based policy learning to learn disjunction of policies within each group. There is a rich set of literature on different variants of boosting that could be adapted to our setting. Suppose, for a training example group, we wish to find the disjunction of policies $H_k(z) = \vee_{i=1}^{k} h_k(z)$ where h_k is the k^{th} rule. Letting TPR_k and FPR_k be the true positive rate and false positive rate respectively after learning the k^{th} rule, we can see that,

$$\begin{aligned} TPR_k &= \Pr_{\mathcal{D}^+}(H_{k-1}(z) \vee h_k(z) = 1) \\ &= TPR_{k-1} + \Pr_{\mathcal{D}^+}(H_{k-1}(z) = 0) \Pr_{\mathcal{D}^+}(h_k(z) = 1 \mid H_{k-1}(z) = 0) \end{aligned}$$

Also, we can write $FPR_k \leq FPR_{k-1} + \Pr_{\mathcal{D}^-}(h_k(z) = 1)$. Say we cap the number of boosting iterations to, say, $N = 3$, then $FPR_k \leq \epsilon$ is ensured if $\Pr_{\mathcal{D}^-}(h_k(z) = 1) \leq \epsilon/N$. Since, $\Pr_{\mathcal{D}^+}(h_k(z) = 1 \mid H_{k-1}(z) = 0)$ can rewritten as $\Pr_{\mathcal{D}^+ \cap I_{k-1}}(h_k(z) = 1)$ where I_{k-1} are the class-R examples incorrectly classified till iteration-$(k-1)$, in iteration-k, we must solve

$$\max_{h_k} \Pr_{\mathcal{D}^+ \cap I_{k-1}}(h_k(z) = 1) \ \text{ s.t. } \ \Pr_{\mathcal{D}^-}(h_k(z) = 1) \leq \epsilon' \ ,$$

where $\epsilon' = \epsilon/N$ where N is the maximum boosting iterations. This is similar to our original problem except that, before iteration-k, we must remove all correctly classified training examples from class-R till iteration-$(k-1)$.

3.5 Interpretability

The structure of a policy could guide an expert towards finding a permanent fix for the problem as follows. First, the expert obtains the symptom associated with the policy. Next, by considering the degraded KPI/s associated with this symptom, the domain expert gets a very good idea about the measurement categories that could get impacted in order of criticality (order depends on the degraded KPI). The feature/s with highest weight (in the classification rule) within a category provides a glimpse into the faults captured by the policy which could be investigated using raw data for different eNodeBs (from the training set) before the remedial action. This way, an expert obtains the key affected measurements, in order of criticality, for a given fault which can be then be used to fix the issue by consultation with the vendor.

4 EVALUATION

Implementation: We have implemented the system on a Linux based high performance computing cluster with 1 TB RAM on each

machine. The implementation is Python-based and uses process-based parallel programming constructs to speed up the training steps. We are also working on an Apache Spark based implementation to evaluate possible gains in computing speed.

We now evaluate AESOP using data collected from an operational cellular network in a major US metropolitan region in the US East coast. The area consisted of around 1800 eNodeBs. Over the course of 3 months of training data, we encountered five different operational environments (software version and configuration).

4.1 Experiment Setup

Measurement data for training: The system learns the policies based on 3 months of history and the policies are updated once a month. The data consisted of around 1100 dimensions (consisting of measurement counters and KPIs) collected every 15 minutes . The time-series data is augmented using a rolling median (as described in Section 3.2) of each dimension. We use two data sets for our results: Section 4.2 is based on policies learned in Jan 2017 and Section 4.3 is based on evaluation of policies learned in April 2017.
Remedial action logs: We show results for a frequently used remedial action taken in the network, specifically rebooting an eNodeB. In other words, we trained and evaluated our system for learning policies that can recommend when an eNodeB can be rebooted to fix a service issue. For the purpose of training, we filtered out all incidents during maintenance hours. Also filtered out were two additional types of reboots: (i) ones that were due to software upgrades, and (ii) ones that were not preceded by any KPI degradation as there are many reboots that are automated and have nothing to do with mitigating service issues. We only focus on cases where the service impact (measured through the aggregate KPI degradation score) is above a minimum threshold as the success of our system depends on effectively detecting service issues with appreciable KPI degradation.
Testing: Once the policies are learned, evaluation/testing is done in a subsequent period (clearly non-overlapping period). Testing is done periodically based on all measurement data collected over the period (we do it once a day but it can be performed once every few hours as well). The testing steps for a measurement instance corresponding to an eNodeB at a given time is as follows:

(1) Normalize the data using the data normalization steps described in Section 3.2.
(2) Associate a policy group based on the software version, configuration template and computed symptoms.
(3) For each policy learned for this group, compute the classification probability $\hat{p}$ of class-R given by (4). We output this measurement instance as class-R if $\hat{p}$ is more than a threshold (obtained through 10-fold cross validation in the learning phase). In standard approaches with ROCs from the K-fold cross-validation [6], one compares $\hat{p}$ to the threshold calculated from the mean-ROC plot; however, due to the sensitivity to false positives in our system, we also associate a confidence to the classification based on the fraction of thresholds (there are 10 threshold once for each cross-validation) exceeded by $\hat{p}$.
(4) For each remedial action from action logs, in the time leading up to the action, find the prediction with maximum confidence.

Table 1: Number of features used and average TPR ($FPR < 0.005\%$) for the best choice of hyper-parameters based on 10-fold cross-validations. We use tptKPI, xaKPI, cdKPI to denote KPIs related to throughput, connection access, and call drops respectively (rather than the exact KPI names). The operational environment tags are anonymized for confidentiality.

Operational Environment (SW Vrsn./Conf.)	Symptom KPIs	Iteration	No. Features	TPR
ST-1	xaKPI-2	0	26	34%
	tptKPI-6	0	60	30%
		1	74	20%
ST-3	cdKPI-1	0	54	80%
ST-5	tptKPI-5 and tptKPI-6	0	62	35%
	xaKPI-2	0	22	36%
	tptKPI-6	0	24	53%
		1	35	23%
		2	65	15%
	cdKPI-1	0	84	29%

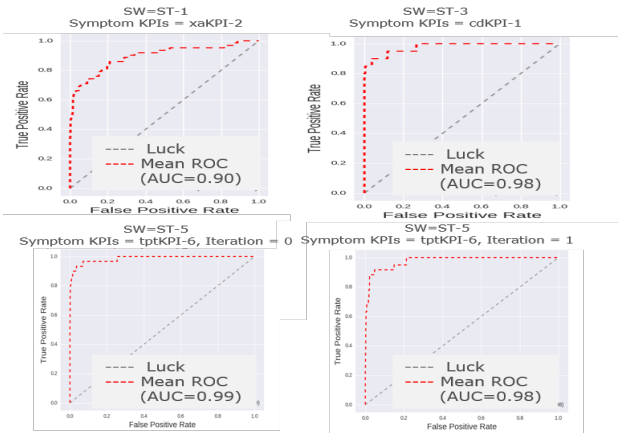


Figure 4: ROC curve for few cases in Table 1

In the following, we first present in Section 4.2 a validation of the individual policies learned by AESOP, and then in Section 4.3 we present results demonstrating the practical usefulness of the pool of policies learned.

4.2 Policy Validation

We first present cross-validation results to understand the effectiveness of our key steps and model. We have three goals: first, we wish to validate that our model outputs policies with good true positive statistics in-spite of very-tight constraints on the false positive rate which is important for practical usefulness of the policies; second, we wish to validate potential gains from boosting in Section 3.4.3; third, we wish to validate whether the best choice of model hyper-parameters provides a significant reduction in the number of counters which is important to make sense out of the extracted features (by a domain expert). All results are based on the 10-fold cross-validation results produced by The best choice of hyper-parameter obtained through a grid search.

In Table 1, we show the 10-fold cross-validation performance of sample policies. In Figure 4 we show some of the ROC curves. We make the following observations:

(1) In Figure 4, we show the ROC curves behind some of the policies in Table 1. Though in most cases, the area under the curve ranged from $0.9 - 0.98$; however, choosing a classification threshold to ensure very low false positive leads to a true positive rate (TPR) in the range $15 - 80\%$. *For our application, even a true-positive rate of, say* 30% *for a learned policy, can lead to significant number of actions that can be predicted using the policy.* Note that the TPR/FPR of each policy is evaluated on a subset of instances that match the operational environment and symptom for that policy.
(2) We have observed that in some of the cases there is benefit to be had from boosting while for many other cases there are no gains. For example, in Table 1, consider TPRs for operational environment ST-5 and symptom with tptKPI-6 (this is a throughput related KPI). The first three iterations of boosting had a cross-validation TPR of $53, 23, 15\%$ respectively. Since the final policy is *OR* of the policies, the expected TPR of the combined policy is $1 - \prod_k (1 - tpr_k) = 0.69$ where tpr_k is the true positive rate of iteration-k. Similarly, for software version/config ST-1 and symptom tptKPI-5, two iterations of TPR 30% and 20% translates to an expected TPR of around 44%.
(3) Most of the policies have just a few tens of features reduced from approximately 2200 dimensions (after feature augmentation to include rolling time-windowed measurements). The two orders of magnitude reduction in dimensions turns out to be very important in making the policies effective as well as interpretable by a domain expert.

4.3 Policy Usefulness in Practice

From an operational perspective, policies are useful if *(i)* they detect a substantial number of service issues that can be recovered through appropriate remedial actions, and *(ii)* they keep the false positives low. We next show how the policies we learn meet these requirements based on some of the test results captured during a two-week period.

In this section, instead of showing the effectiveness of individual policies, we show results based on the collection of learned policies. The final recommendation is a reboot if *any* of the policies in the collection recommends a reboot with high confidence. In our results, we select policies to be part of the collection if cross-validation performance during training gives a true positive rate above a minimum threshold (chosen as 25%).

In order to have meaningful TPR and FPR, we choose test candidates as those measurement instances where there is a matching operational environment and symptom for *any* policy in the collection. Also, from our test candidates, we filter out reboots that were not effective as we are interested in detecting (i.e. predicting) effective reboots. Effectiveness of a reboot is based on comparing aggregate primary KPI degradations in fixed length time windows before and after each reboot. In the rest of this section, we simply say reboot to mean effective reboot.

Prediction of remedial actions: Figure 5 depicts the prediction confidence of the candidate reboots from action logs. As we

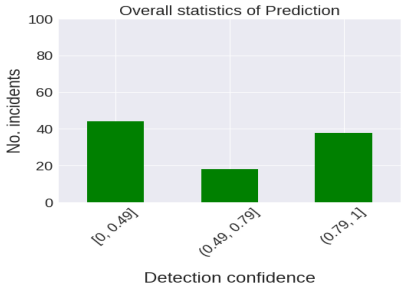
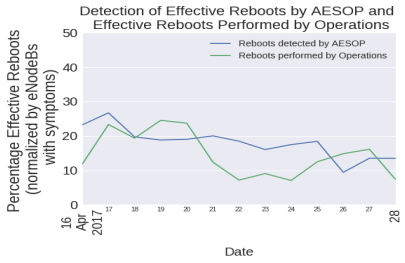
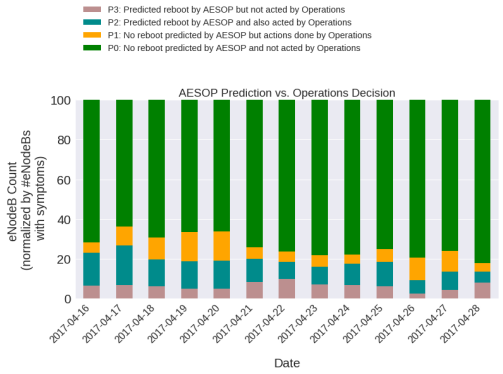


Figure 5: Statistics of detection confidence for effective reboots among test-candidates.



(a)



(b)

Figure 6: (a) Time series of number of effective reboots detected by reboot and performed by operations (b) Time-series showing breakdown of effective reboots detected by AESOP.

can see, our system detected (i.e. correctly predicted) around 40% of these reboots with high confidence (0.8 or above) in the time leading up to the eNodeB reboot. This demonstrates AESOP's potential for automatically learning policies that can be used in automation systems such as ECOMP[1] for achieving operational efficiencies.

Comparison of reboots detected by AESOP with those performed by operations' : Figure 6a compares number of reboots detected by AESOP with those performed by operations. In Figure 6b, as a daily time-series, we provide breakdown of comparison between predictions by AESOP and decision by Operations. The plot can be interpreted as follows:

- The area P0 represents percentage of instances where neither AESOP detected a reboot nor any reboot was performed by

operations among all instances where there was a symptom. This shows that, our system can effectively detect cases where no reboot is required.

- The area P1 represents percentage of instances where AESOP did not detect a reboot with high confidence but operation carried out a reboot. The gap could be attributed to the fact that our models are tuned for very low false positive as a design decision.
- The area P2 represents percentage of instances where AESOP detected reboot and a reboot was carried out by operations in the subsequent 24 hrs.
- The area P3 simply represents percentage of instances where AESOP detected reboot but a reboot was not carried out in the next 24 hrs.

5 RELATED WORK

There is a a considerable amount of work on designing data driven analytical tools to monitor and detect anomalies in networks [2, 9, 10, 18, 20], and also to detect service impact due to software upgrades [11, 12]. In the context of networks, there has also been some work on developing probabilistic models [14] for root cause analysis, designing root cause analytics platforms [19], and automation of system problem diagnosis [5, 21]. While all of these and similar works develop data driven methodologies to aid the operations personnel, our goal is to automate policy learning for operations.

In recent years, the broad scope of designing data driven learning systems for network management and control has received attention: [13] develops a system for automatically learning control decisions for resource management in data centers, [3] proposes a system for proactively predicting disk failures, [17] designs a system for automatically mining syslogs to detect anomalies in application program control-flow, and [16] proposes a learning based system for detecting and monitoring service issues. The main difference with our work is that we design a system that accounts for some of the unique operational challenges in large networks by fusing domain knowledge with machine learning.

6 CONCLUSIONS

In this paper we presented AESOP, a data driven system for automatic policy learning for triggering remedial actions in networks. To the best of our knowledge, this is the first work on automatic policy learning for network operations that accounts for the challenges specific to networking in a systematic manner. While we demonstrated the effectiveness of the policies learned, we wish to emphasize that we are working on evolving our policy learning framework to capture more complex symptoms and fault conditions so as to have wider applicability of the policies. The design principles of AESOP are applicable to any software centric networking and distributed Systems, including those which leverage virtual network functions.

ACKNOWLEDGMENTS

The authors would like to sincerely thank Sanjeev Ahuja from AT&T's operations team for sharing his insights and experience that helped our design.

REFERENCES

[1] 2016. ECOMP (Enhanced Control, Orchestration, Management & Policy) Architecture White Paper. http://about.att.com/content/dam/snrdocs/ecomp.pdf. (2016).
[2] Paul Barford, Jeffery Kline, David Plonka, and Amos Ron. 2002. A signal analysis of network traffic anomalies. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 71–82.
[3] Mirela Madalina Botezatu, Ioana Giurgiu, Jasmina Bogojeska, and Dorothea Wiesmann. 2016. Predicting Disk Replacement Towards Reliable Data Centers. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 39–48.
[4] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.
[5] Ira Cohen, Jeffrey S Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons. 2004. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control.. In *OSDI*, Vol. 4. 16–16.
[6] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.
[7] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. 2000. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics* 28, 2 (2000), 337–407.
[8] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 58–72.
[9] Yu Gu, Andrew McCallum, and Don Towsley. 2005. Detecting anomalies in network traffic using maximum entropy estimation. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 32–32.
[10] Anukool Lakhina, Mark Crovella, and Christophe Diot. 2004. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, Vol. 34. ACM, 219–230.
[11] Ajay Mahimkar, Zihui Ge, Jennifer Yates, Chris Hristov, Vincent Cordaro, Shane Smith, Jing Xu, and Mark Stockert. 2013. Robust Assessment of Changes in Cellular Networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '13)*. ACM, New York, NY, USA, 175–186.
[12] Ajay Anil Mahimkar, Han Hee Song, Zihui Ge, Aman Shaikh, Jia Wang, Jennifer Yates, Yin Zhang, and Joanne Emmons. 2010. Detecting the Performance Impact of Upgrades in Large Operational Networks. In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM '10)*. ACM, New York, NY, USA, 303–314.
[13] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource Management with Deep Reinforcement Learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, New York, NY, USA, 50–56.
[14] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, and Christophe Diot. 2004. Characterization of failures in an IP backbone. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, Vol. 4. IEEE, 2307–2317.
[15] Nicolai Meinshausen and Peter Bühlmann. 2010. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72, 4 (2010), 417–473.
[16] Vinod Nair, Ameya Raul, Shwetabh Khanduja, Vikas Bahirwani, Qihong Shao, Sundararajan Sellamanickam, Sathiya Keerthi, Steve Herbert, and Sudheer Dhulipalla. 2015. Learning a Hierarchical Monitoring System for Detecting and Diagnosing Service Issues. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, New York, NY, USA, 2029–2038.
[17] Animesh Nandi, Atri Mandal, Shubham Atreja, Gargi B. Dasgupta, and Subhrajit Bhattacharya. 2016. Anomaly Detection Using Program Control Flow Graph Mining From Execution Logs. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 215–224.
[18] Marina Thottan and Chuanyi Ji. 2003. Anomaly detection in IP networks. *IEEE Transactions on signal processing* 51, 8 (2003), 2191–2204.
[19] He Yan, Lee Breslau, Zihui Ge, Dan Massey, Dan Pei, and Jennifer Yates. 2012. G-RCA: A Generic Root Cause Analysis Platform for Service Quality Management in Large IP Networks. *IEEE/ACM Trans. Netw.* 20, 6 (Dec. 2012), 1734–1747.
[20] He Yan, Ashley Flavel, Zihui Ge, Alexandre Gerber, Daniel Massey, Christos Papadopoulos, Hiren Shah, and Jennifer Yates. 2012. Argus: End-to-end service anomaly detection and localization from an ISP's point of view.. In *INFOCOM*. IEEE, 2756–2760.
[21] Steve et. al. Zhang. 2005. Ensembles of models for automated diagnosis of system performance problems. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*. IEEE, 644–653.