

SYSTEM BEHAVIOR ANALYSIS BY MACHINE LEARNING

Yuncheng Li*

Department of Computer Science
University of Rochester
yli@cs.rochester.edu

ABSTRACT

This survey presents several system instances and how Machine Learning are involved for system design, development, evaluation and test. To facilitate the explanation for these analysis examples, I provided some Machine Learning background after the introduction section.

Index Terms— System Behavior, Machine Learning, Principle Components Analysis, Feature Engineering

1. INTRODUCTION

System anomaly detection, dynamic resource allocation and source code debugging are crucial in today's computing system development. System anomalies include system failure, abnormal latency, network congests and so on. On one hand, these anomalies could undermine developer's reputation. On the other hand, incidents of these anomalies indicate system problems. For example, a sudden traffic peak may indicate a DOS attack risk. As another research challenge, dynamic resource allocation involves system state identification. During different system states, resources demanding are different and if we could allocate resources accordingly, maximum utilization would be achieved. In another scenario, as system scale grows, source code debugging becomes unrealistic with just manual labor.

One of the main issues for these system behavior analysis is scalability. As a system manager, one may want to detect duplicated bug reports and submit clustered bug reports to developer with different importance priorities. As a web server administrator, one may want to predict system failure, so that it could be eliminated with proper action. As a large scale software developer, one may want to get a summation from the console logs output, rather than read the console logs directly.

In the perspective of data analytic, there are several challenges to deal with this kind of system behavior related data. First of all, the raw data are badly organized with arbitrary formats. Different developer has its own genius way to output logs or reports. Secondly, with system scale growing, data

are always in high dimensions and large scale. In addition, data are easily corrupted with noise and the problems include missing data, weakly labeled and unbalanced data instances.

To perform general system behavior analysis, several principles are important to deal with these challenges. First of all, the analysis should be data driven, rather than problem specific. One could write some shell scripts to deal with its own tiny data, but what is important here is to solve very general problems with wide range application. For this reason, pattern recognition and data mining technology are required.

2. MACHINE LEARNING BACKGROUND

Several specific analysis techniques used in this survey are introduced in this section. To make this report concentrate to its main topic, all these introductions are just scratching the surface.

Principle Components Analysis The idea behind Principle Components Analysis(PCA) is to perform dimension reduction. Formally, we have data instances $\mathbf{Y}_i = (y_i^1, y_i^2, \dots, y_i^m), i = 1, 2, \dots, n$, which are called features. Each feature is an m-array and we have n instances of this kind of features. With PCA, we want to find another set of m-array $\mathbf{W}_k = (w_k^1, w_k^2, \dots, w_k^m), k = 1, 2, \dots, K$, such that it could be used to linearly reconstruct original features $\mathbf{Y}_i = \sum_{k=1}^K a_i^k * W_k + \varepsilon_i$. This could be represented by matrix multiplication $\mathbf{Y} = \mathbf{A} * \mathbf{W}$. Each row of \mathbf{W} is called principle components and each row of \mathbf{A} is the reconstruct coefficients. There are two requirements for this set of m-array \mathbf{W}_k ,

1. $K < m$, so that each feature \mathbf{Y}_i could be represented by the linear reconstruction coefficients $\mathbf{A}_i = (a_i^1, a_i^2, \dots, a_i^K)$ with lower dimensions ($K < m$).
2. Total Error term $\sum |\varepsilon_i|^2$ should be as small as possible.

With normal data, we could establish a set of principle components, which is a low dimension representation

*This is the author's survey report for the CSC456 Operating System Course

of normal system state. To evaluate whether a new data instance is normal or not, we could reconstruct it with the set of principle components and the reconstruction error is the indication of whether its normal or not. This procedure is a standard outlier detection with PCA.

Minimum Editing Distance The goal for Minimum Editing Distance(MED) is to compare a pair of arbitrary strings. A set of editing operations are defined, with which one could transform one string to another. Then, the MED is the transformation with minimum number of steps. Therefore, the MED defines a similarity metric between two strings.

This definition of MED could be extended to higher level strings, such as functions or modules in a call stack. In addition, we could enforce different weights for different kinds of operation. For example, we may penalty more deletion than insertion in certain circumstance.

Logistic Model Logistic Model is a kind of probability prediction model. $P(Y|X)$, the probability distribution of dependent variable given the independent variables, is modeled by logistic function $y = \frac{1}{1+e^{-H \cdot X}}$. Logistic Model have several good advantages. First of all, it's easy to train and good generality, since in essence it's a linear model. Secondly, it has little assumption about the independence in features, since it's a conditional model. With this property, we could freely use any kinds of rich features with maximum expressive power, but may have hairy dependence among features. In addition, Native way to extend to multi-class prediction.

Reduced Model Model complexity grows with its number of parameters and if two models have similar performance, we prefer the model with less parameters. Model reduction is an effective way to eliminate unnecessary parameters by enforce some constraints on the parameters of the original full model.

Precision and Recall This is one of the standard way for classifier evaluation. To evaluate a binary classifier ($Y_{predict} = h(X)$), we perform prediction on test data (X_{test}, Y_{test}) which yields four kinds of results. If we have $Y_{predict} = 1$ and its corresponding $Y_{test} = 1$, it's called true positive(TP). Similarly, we could define true negative(TN), false positive (FP), false negative (FN). After counting the number of each type of results, we could calculate precision and recall as follows,

$$\begin{cases} Precision = \frac{TP}{TP+FP} \\ Recall = \frac{TP}{TP+FN} \end{cases} \quad (1)$$

Intuitively, *Recall* is a measure of sensitivity (*Precision* could be intuitively understood by its name). It's

a common sense that high sensitivity may lead to low precision and vice verse. We prefer models with more precision at the same level of sensitivity.

Bag of words We could view Bag of Words (BoW) as a way to quantize continuous feature space. For example, although color space is continuous in terms of optics, human eyes could only distinguish a certain number of colors, such as *red, orange, blue, black, green, yellow, ...*. One way to describe an image is its histogram of different colors.

BoW could be intuitively understood in the following way. With a continuous feature space, we first quantize it into several *words* and then count the number of words in a certain "*document*" to get a histogram description.

There are several building blocks for Machine Learning Analysis and I will introduce each analysis example accordingly.

- **Raw Data**, which are usually not well structured, such as console logs from arbitrary program
- **Feature Engineering**, refining raw data by removing unnecessary information and transforming data representation into a form that could be used by general data model
- **General Data Model**, such as Tree Based Model (CART), Principal Components Analysis (PCA), Logistic Model ...

3. ANALYSIS EXAMPLES

In this section, several analysis examples are listed to demonstrate the usage of machine learning techniques for system behavior analysis.

3.1. Windows Error Reporting System[1]

As an instance of modern operating system, Microsoft Windows suffers from system debug and test. There are no way to perform full debug and test before release product. Therefore, error report from user is crucial to improve system performance and consequently company's revenue. With the collected error report, developer could find the problem and fix it. However, it's wasting resources to assign developers to fix the same problem over and over again.

Therefore, before submitting the error report to developers, detecting and removing duplicated reports will be of great help. Here comes the first example of data mining in system behavior analysis.

Raw Data Each bug reports contains several entries, which are useful for developer to fix the problem. These

Module	Function	Offset
kernel32	ByteCallback	0x3
kernel32	WideCharExpand	0x2
kernel32	MultiByteToWideChar	0x9
A3DHF8Q	---	0x3820523
A3DHF8Q	---	0x3723952
A3DHF8Q	---	0x3945323
kernel32	ProcUserText	0x4
user32	TextDecode	0x4096
user32	ReadDialog	0x4096
user32	OpenDialog	0x4096
ntdll	RtlThreadStart	0x0
ntdll	RtlInitThreadThunk	0x0

Fig. 1. A Windows Operating System Callstack example

entries include (i) Failure Type, such as segmentation fault and deadlocks; (ii) Process Name; (iii) Exception code, such as 0x c0 00 00 00 05; (iv) Offending callstack, Fig. 1 is an example of Callstack.

Features Features are extracted to make it useful for data model. The features used in [1] include (i) Indicator Function for *Event Type, Process Name and Exception Code*; (ii) Extended Minimum Editing Distance for pairwise Callstack comparison ($O(\min(m, n))$ space complexity). The MED editing operations include insertions, deletions and substitution and its editing target includes letters, frames and callstack groups.

Data Model *Logistic Model* as mentioned above is used for pairwise similarity prediction. The input are several sets of error reports. Each set of reports are labeled by developers as the same problem, and the output is a *prediction model* that could decide whether a pair of error report should be grouped together. In addition, [1] utilize different sets of reduced model for model evaluation.

Results Fig. 2 demonstrate the precision-recall curves of different similarity prediction model. As is mentioned above that we prefer higher precision in the same level of sensitivity, the proposed method outperforms previous works. In addition, when the full model are reduced, there are not perceptible performance loss.

3.2. Web server performance metric[2]

In large scale web server system, any system errors are fatal for the system’s reputation. Hence, an effective way to predict and avoid web server failure is crucial. To achieve failure prediction and avoidance, the system should be able to alarm emerging failure, associate current state with historic logs and identify causes of this failure.

In addition, system state identification is valuable for dynamic resource allocation. Some state may demand more

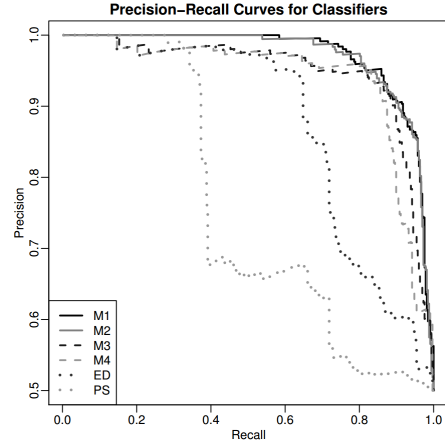


Fig. 2. Precision and Recall Performance. M1 is the full *Logistic Model*, M2 - M4 is the reduced *Logistic Model* with different levels of model reduction. ED and PS are previous works by other researchers.

bandwidth and some other states more CPU cycles or network traffic. These kind of information is useful to achieve maximum resource utilization.

In [2], the author took advantages of some standard system performance metrics such as CPU overload, disk IO and network traffic, to achieve several system behavior related analysis,

- *Clustering* for identifying distinct system states with Kmeans or Tree Based Model
- *Correlation Analysis* to find which set of attributes always comes with a certain type of failure, which is an indication of failure causes
- *Baseline analysis* and detect anomaly that far from baseline

3.3. Console logs with source code[3]

The last example deals with data mining the console logs. One effective way for programmer to debug their code is to print program state to console, especially among the error handling code. However, as the scale of software system grows, it becomes infeasible to analysis the console output manually, so an effective way to analyze console logs to generate a concise summation, becomes increasingly important.

Although the console logs summation sounds fantastic to all programmers, it’s not very well defined. One of the most difficult problem is how to deal with the unstructured data. Each programmer has his or her genius way to print out console logs, but the analysis should be able to output its uniform representation. One thing worth noting is that the system is not attempting to semantically understand the logs output, but

Table 1: Detection Precision

Anomalous Event	Events	Detected
Empty packet	1	1
Failed at the beginning, no block written	20	20
WriteBlock received java.io.IOException	39	38
After delete, namenode not updated	3	3
Written block belongs to no file	3	3
PendingReplicationMonitor timed out	15	1
Redundant addStoredBlock request	12	1
Replicate then immediately delete	6	2

Table 2: False Positives

False Positive Type	False Alarm
Over replicating (actually due to client request)	11
Super hot block	1
Unknown reasons	2

Fig. 3. Manual Labeled and Detection Results comparison

just construct a uniform summation (to some extent) from the unstructured logs and the programmers still needs to solve the real programming problems.

As is mentioned above, Feature Engineering is crucial to deal with this kind of badly constructed data. In the [3], source code and its console logs are analyzed together to extract useful logs field, which are subsequently quantized by Bag of Words (BoW) procedure.

Briefly, the feature extraction procedure is listed as follows,

- Extract Message types and message variable from source code, such as *sent_packet 196.168.3.1*
- identifying *significant* message types that have enough *variations* and grouping message values, by clustering in the feature space
- Generate *bag of words* feature, which is a histogram of message groups

With the extracted BoW features, [3] use Principle Components Analysis to perform anomaly detection. The detected anomaly could be used as a summation of the console logs and the experiment result is listed in Table. 3, which demonstrates promising anomaly detection performance.

4. SUMMARY

In this survey, some Machine Learning techniques are presented as a basis to understanding the listed analysis examples. As is presented in the three referred papers, Machine Learning could be used for system design, development, evaluation and improvement. The main challenge, though, is to fully understand the system behavior related data and extracting efficient features from it, which calls for solid knowledge and insight in both Systems and Machine Learning. In a word,

using Machine Learning to perform System analysis may be a new striving pushing force for both Machine Learning and System communities.

5. REFERENCES

- [1] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan, “Detecting large-scale system problems by mining console logs,” in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, New York, NY, USA, 2009, SOSP ’09, pp. 117–132, ACM.
- [2] Kevin Bartz, Jack W. Stokes, John C. Platt, Ryan Kivett, David Grant, Silviu Calinoiu, and Gretchen Loihle, “Finding similar failures using callstack similarity,” in *Proceedings of the Third conference on Tackling computer systems problems with machine learning techniques*, Berkeley, CA, USA, 2008, SysML’08, pp. 1–1, USENIX Association.
- [3] Songyun Duan and Shivnath Babu, “Empirical comparison of techniques for automated failure diagnosis,” in *Proceedings of the Third conference on Tackling computer systems problems with machine learning techniques*, Berkeley, CA, USA, 2008, SysML’08, pp. 2–2, USENIX Association.