Efficient end-to-end monitoring for fault management in distributed systems

Dawei Feng

UNIVERSITE PARIS-SUD

ÉCOLE DOCTORALE : *Informatique de Paris-Sud*
Laboratoire de *Rechercher en Informatique*

DISCIPLINE Informatique

THÈSE DE DOCTORAT

soutenue le 27/03/2014

par

Dawei FENG

Efficient End-to-End Monitoring for Fault Management in Distributed systems

Directeur de thèse :	Cécile Germain-Renaud	Directrice de thèse (Université Paris Sud, LRI/TAO)
Composition du jury :		
Président du jury :	Joffroy Beauquier	Président du jury (Université Paris-Sud, LRI/ParSys)
Rapporteurs :	Lorenza Saitta	Rapporteur (Università del Piemonte Orientale)
	Johan Montagnat	Rapporteur (CNRS)
Examinateurs :	Michèle Sebag	Examinatrice (CNRS)
	Xiangliang Zhang	Examinatrice (King Abdullah University of Science & Technology)
	Irina Rish	Examinatrice (IBM T. J. Watson Research Center)

Abstract

In this dissertation, we present our work on fault management in distributed systems, with motivating application roots in monitoring fault and abrupt change of large computing systems like the grid and the cloud. Instead of building a complete a priori knowledge of the software and hardware infrastructures as in conventional detection or diagnosis methods, we propose to use appropriate techniques to perform end-to-end monitoring for such large scale systems, leaving the inaccessible details of involved components in a black box.

For the fault monitoring of a distributed system, we first model this probe-based application as a static collaborative prediction (CP) task, and experimentally demonstrate the effectiveness of CP methods by using the max margin matrix factorization method. We further introduce active learning to the CP framework and exhibit its critical advantage in dealing with highly imbalanced data, which is specially useful for identifying the minority fault class.

Further we extend the static fault monitoring to the sequential case by proposing the sequential matrix factorization (SMF) method. SMF takes a sequence of partially observed matrices as input, and produces predictions with information both from the current and history time windows. Active learning is also employed to SMF, such that the highly imbalanced data can be coped with properly. In addition to the sequential methods, a smoothing action taken on the estimation sequence has shown to be a practically useful trick for enhancing sequential prediction performance.

Since the stationary assumption employed in the static and sequential fault monitoring becomes unrealistic in the presence of abrupt changes, we propose a semi-supervised online change detection (SSOCD) framework to detect intended changes in time series data. In this way, the static model of the system can be recomputed once an abrupt change is detected. In SSOCD, an unsupervised offline method is proposed to analyze a sample data series. The change points thus detected are used to train a supervised online model, which gives online decision about whether there is a change presented in the arriving data sequence. State-of-the-art change detection methods are employed to demonstrate the usefulness of the framework.

All presented work is verified on real-world datasets. Specifically, the fault monitoring experiments are conducted on a dataset collected from the Biomed grid infrastructure within the European Grid Initiative, and the abrupt change detection framework is verified on a

dataset concerning the performance change of an online site with large amount of traffic.

Acknowledgements

First and foremost, I want to thank my PhD. supervisor, Cecile Germain-Renaud. Without her help and dedicated assistant throughout my three and half years' Phd. time, this dissertation would have never been accomplished. I would like to express my immense gratitude for her support and understanding over these past 42 months.

I would also like to thank Michele Sebag. For all the discussions she had enlightened me, and the comments as well as suggestions for my research. It has been a real pleasant experience to work with her.

I would also like to show gratitude to my committee: Joffroy Beauquier, Lorenza Saitta, Johan Montagnat, Xiangliang Zhang and Irina Rish. Thank Lorenza and Johan for reviewing my manuscript, giving comments and suggestions on it. This will be very helpful for my future career. Special thank to Xiangliang Zhang, for the pleasant journey in KAUST as well as all the help she gave on my research. Many thanks to the TAO team member. I thank Julien NAUROY, Lovro Ilijasic, Riad AKROUR and Tristan Glatard at INSA Lyon. They helped me a lot on my research in these years.

The accomplishment of my dissertation required more than academic support. I thank my friends for all the assistants they gave and the pleasant memories we had together: Yangbin Tang, Weijia Wang, Wusheng Jiang, Yufei Han, Weihua Yang, Bo Li, Jingyi Bin, Ning Chu, Jianqiang Chen, Guohua Zhang, Jialin Liu, Jonathan, and many other people. I would also like to express special thank to Yifan Yang and Chuan Xu, for translating the French abstract of my dissertation.

Most importantly, this dissertation is dedicated to my family, especially my parents. Nothing would have come true without their support. All I have and will accomplish are only possible because of their love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We depend on computer systems that are not dependable: the pervasiveness of real-world large scale distributed systems in modern Information Technology infrastructures and usage is now well established, and such systems have been characterized decades ago as those where "the failure of a computer you didn't even know existed can render your own computer unusable".

Computer Science research has worked on fault management for such systems since very long, with two main directions: discovering faults, and/or coping with them. The main change associated with the advent of truly massively distributed systems with complex structures is that rich monitoring information has become available. While full knowledge, and the very concept, of the state of a distributed system remains infeasible for fundamental reasons, the availability of equally massive information has make possible to consider estimating elements of the system state as a realistic goal. Specifically some of fault management questions can be re-casted as *inference* problems.

Moreover, estimates in the area of fault discovery in turn provide useful information for the next task of coping with them, as such systems are normally highly redundant and heavily supervised; thus in many cases, alternatives to the faulty components can be proposed; in these cases, a well organized fault management system will conceal the hardware and software dysfunctions and will be capable to provide a transparent service that is a crucial ingredient of Quality of Experience. Irrecuperable faults, on the other hand, must be signaled as fast as possible to the human of automatic supervision. Overall, this amounts to re-evaluate the role of monitoring in fault management, and to consider

the monitoring task as an inference in a spatio-temporal domain.

The focus of this dissertation is on the specific aspect of fault monitoring which is fault discovery. More precisely, we consider two modalities of faulty behavior: on the one hand, service availability considered in a binary setting - that is, the service is either available or not-, and on the other hand performance of the service, where a quantitative drop in performance defines a fault. In the first case, we model the components of a large distributed system as the row- or column- entities in a system status matrix, while their interconnection status are entries in this matrix. The main task of fault monitoring is therefore to predict the whole status matrix with a partially observed entry set as input. In the performance monitoring case, our goal is to detect change of a service performance in an online manner. Different types of performance measurements describing the real-time behavior of a service are collected and analyzed online, decisions about whether there is a change and when the change took place are made from the performance data stream.

The final goal of fault discovery is to improve system availability and reliability in terms of providing users or the higher level of the monitoring system with accurate and meaningful information on existing or possible faults. The most straightforward approach is then detection and/or diagnosis, where a detailed internal model of the system is exploited to pinpoint the faulty components or at least the possibilities of faults. The root causes of the faults can be diagnosed through various techniques like statistical inference, log-based causality analysis or deterministic replay. This fault diagnosis can be seen as the process of recognizing the most likely explanation for the symptoms based on some causal and effect models among the propositions of interest in the problem domain.

While such approaches maximize the usefulness of monitoring data, they face some potentially significant practical limitations. The first is simply scalability. Moreover, assuming that a decent model of the system is available may often be unrealistic. As a consequence, this work formulates the fault discovery problem in a black-box fashion: only (possibly faulty) end-to-end connections are visible.

Within this framework, in order to be realistic, inference has to address two specific difficulties. Firstly, strongly imbalanced distributions must be assumed, as faults are hopefully much less represented than nominal behavior; this belongs to the spatial aspect of inference. Second, in the temporal domain, one cannot assume that measurements could be kept fully up-to-date, as these systems are highly dynamic environments.

Fortunately, the same strategy has been successful in various context to address both imbalanced distributions and noisy information: Active Learning selects most-informative samples in order to best improve the prediction accuracy. On the other hand, and always with realism in mind, Active Learning has the drawbacks to slow down the fault discovery process and to make it more complicated, thus more fault-prone itself. A transversal goal of this work is thus to evaluate the contribution of the Active Learning ingredient in the fault inference methods that we propose.

The motivating application of this thesis is the fault management of large computing systems like the grid and the cloud. Grids have become to be considered somehow old-fashioned, thus a few words about their relevance might be necessary. The specific technologies that have been used to build grids in the 2000's have of course been superseded by cloud-related ones. However, the essential paradigm of grid is organized sharing: safely and fairly federating hardware, software and data resources from multiple independent providers. Thus grids exemplify both the physical problems of worldwide scale systems, and the additional and major issues associated with a multi-owned multi-operated system.

Based on above motivations, this thesis is mainly devoted to monitor system fault and performance change in large scale system with assistant technologies from the machine learning field. The main contributions can be summarized as follows:

1. For the fault monitoring of components interconnection in a distributed system, we first model this probe-based application as a collaborative prediction (CP) task, and use the max margin matrix factorization to experimentally show the effectiveness of collaborative prediction methods. We further introduce active learning to the existed CP methods, and exhibit its critical advantage in the fault monitoring application[51, 50].
2. A second contribution concerns the extension of the static fault monitoring to the sequential case. The proposed sequential matrix factorization (SMF) method, takes a sequence of partially observed matrices as input, and produces predictions with information both from the current and history time windows. The extension of SMF with active learning (SMFA) is also proposed in the sequential case, where its ability to make accomplished predictions is exhibited in a highly imbalanced dataset. In addition to the sequential methods, a smoothing action taken on the estimation sequence of each algorithm has shown to be a practical useful trick which brings better prediction

performance.

3. The third contribution focuses on the online change detection in time series data. A semi-supervised change detection framework is proposed to detect the preferred changes in an online data sequence. Labels of preferred changes (significant changes) are firstly learned by offline segmentation algorithm, and online change detection algorithm is trained on the learned label set in a supervised way. Decision about whether there is a change presented in the arriving data sequence is finally made by the learned online change detection algorithm.
4. Last but not least, we have verified the proposed methods on real-world datasets. More exactly, for the fault monitoring, we have verified the proposed collaborative methods on a dataset collected from the Biomed grid infrastructure within the European Grid Initiative, i.e., predicting the functional availability between computing elements (CE) and storage elements (SE) in the Biomed grid. For the performance monitoring, we have verified the proposed method on a dataset collected from an online site with large amount of user traffic, i.e., uncovering the change patterns of a site's performance.

The manuscript of this dissertation is organized as follows: contents mainly concerned with fault monitoring are described in Chapter 2 and 3. More exactly, Chapter 2 reviews the state of the art research on fault management in distributed systems. Chapter 3 discusses our work on the prediction of a partially observed status matrix in a static way, under the assumption that the system status is static in the tested time window. Additionally, efforts have been put into dealing with the more realistic sequential fault monitoring problem in Chapter 4. Specifically, the static monitoring described in Chapter 3 is extended to a sequential formulation, such that predictions are produced based on a sequence of partially observed matrices. Chapter 5 discusses the problem of performance change detection of an online service with large traffic. A semi-supervised framework which combines the offline segmentation and online change detection techniques is proposed and validated on a real-world online site. Some conclusions and future perspectives are presented in Chapter 6.

Chapter 2

Fault Monitoring

2.1 Introduction

Isolating the end users of an Information Technology system from the hardware and software malfunctions is recognized as a critical component of Quality of Experience [134]. To give only one example, the crash of the Amazon Cloud [25] highlighted the importance of timely discovery of failures: a local, limited error may result in a global catastrophe. More generally, fault management of distributed systems has conventionally been of vital importance both to the management of systems and to the service level agreement. It has attracted much attention both from the academic and industrial fields with the extensive deployments of large scale systems like clouds and grids in recent years [146] [60] [180] [102] [83] [38] [76] [85] [165] [42]. From the system point of view, a well operated fault management system is the basic support of its services, and can be widely used in applications like resource scheduling and allocation, performance optimizing for services and middleware, security monitoring.

In large distributed systems, faults are not only unavoidable, but frequent, and a major goal is to cope with routine faults. As Autonomic Computing (AC) [88] has been a major conceptual framework for designing and analyzing truly large distributed systems, it is worth briefly positioning this work in the AC context. The AC approach at large is based on the so-called MAPE-K loop: monitor/analyze/plan/execute, all organized around a Knowledge component (Fig. 2-1). Fundamental results [61] limit the accuracy of this knowledge in principle, and its approximations cannot be built a priori, but must be inferred from behavioral data: a significant part of the software infrastructure of large scale distributed

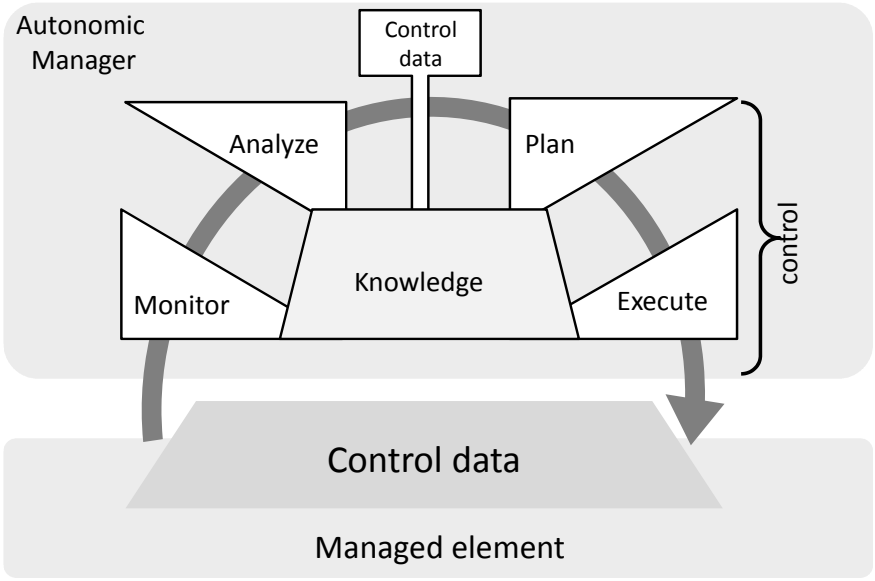


Figure 2-1: MAPE-K loop, autonomic computing

systems, collects information (monitoring) that will be exploited to discover (knowledge) if, where, or when the system is faulty.

This chapter discusses more precisely which kind of knowledge is actually reachable when faults are concerned, in order to set up *fault inference* as a realistic objective for a fault monitoring strategy. In particular, we report on our attempts to model the faults of a subsection of a real-world distributed system, EGI (the European Grid Initiative), and we motivate the selection of end-to-end probing as the class of monitoring techniques.

2.2 Fault management for distributed middleware

Fault tolerance and fault management are two widely used techniques in developing a robust and secure distributed system against faults. While fault tolerance strives for tolerating system faults (either hardware or software) and continuing its intended service at a contracted level, instead of failing down completely, fault management focuses on improving availability and reliability in terms of providing users with accurate and informative information of existing faults.

As the complexity of distributed computing systems increases, various efforts have been put into the filed of fault management. [143] attempts to build a proactive prediction

and control system for large clusters. Event logs concerning various system reliability, availability and serviceability are collected from a relatively large (350 node) cluster, and used for system performance prediction. Time-series prediction, Bayesian network models, and rule based classification algorithms are evaluated on the dataset, showing a feasible result of system performance predicting. [41] proposes a self-healing method for autonomous detection and handling of operational incidents in scientific work-flow activities. Incident degrees of work-flow activities are measured by metrics like long-tail effect [37] (a commonly encountered problem for users who have to wait for a long time to retrieve the last few pieces of their computations), application efficiency, data transfer issues, and site-specific problems. A rule based method which captures the correlations between incident levels is applied for selecting healing actions. In the aim of improving grid application performances, [104] introduces a stochastic model to describe the behavior of a complex grid workload management system. Model parameters are learned from historical grid activity traces and used for optimizing a simple job resubmission strategy. As shown in the paper, the proposed model is able to provide a quantitative results for enhancing job submission performance and a quantified impact for measuring faults and outliers on grid operations. [180] proposes to organize fault management in distributed systems into mainly three steps, i.e., fault detection, fault diagnosis and evidence generation. More exactly, the first step is to monitor the system behaviors and check the observation against predefined system specifications, i.e., discover if any of the components is faulty. While the second step which exhibits all faulty components and the root cause of the fault is triggered once a fault in step one is detected. Finally, evidences which describe the assertions drawn from system diagnosis in detail are generated to system administrators.

In the following, we will use a different terminology: monitoring is ambiguous, as it can describe the low-level data acquisition as well as the exploitation of the resulting information, which involves the whole loop MAPE-K loop in the Autonomic context, or the diagnosis and high-level reporting. We choose to use *monitoring* in the second sense, and reserve *acquisition* for the restricted sense.

2.2.1 Data acquisition

Before applying any fault detection algorithm we need firstly to collect data from the system. There are generally two classes of approaches depending on whether a method actively

performs measurements (e.g., using benchmarking algorithms) or passively collects performance data from platform specific facilities [39] [173] [12]: **passive data collection** by injecting monitoring scripts into the system to collect runtime data inside the system, and **active data collection** by launching testing probes for inspection from outside.

In the passive data collection, a special designed device, like Sniffer in the network traffic monitoring [127] or designed code woven in software via AOP (Aspect Orientated Programming) in the software behavior monitoring [35], is used to watch and collect system information. The data is collected periodically or triggered by some predefined events and alarms are bubbled up to the management system when certain conditions are satisfied or threshold exceeded. Many approaches can be categorized into the passive class. Commands such as *top*(CPU utilization information of each process), *vmstat* (memory and cpu utilization statistics) are commonly used passive monitoring tools in Unix systems. More sophisticated implementations are designed for application specific problem. [23] introduces a passive DNS analysis approach, EXPOSURE, for detecting domain names that are involved in malicious activity. In contrast to active DNS monitoring in [74], which uses probes for detecting suspected malicious domains, the passive method in [23] is stealthy and does not trigger any extra malicious activity in order to obtain information about the domain. This contributes to an advantage that the attackers have no means to block or hinder the performed analysis (in contrast to active approaches). [66] uses a centralized manager to identify the problem by correlating the received alarms, which are collected at runtime using injected codes in each monitored device. Ganglia [116] is a scalable distributed monitoring system for high performance computing system such as clusters, grids. It provides the user with a remote live view or historical statistics (such as CPU load averages or network utilization) for all machines being monitored. More passive monitoring based applications can be found in [173].

Passive data collection can provide an extremely detailed view of the system, e.g., runtime function call tree or individual packets passing through a monitored network device, thus it is more appropriate for supporting fine-grained operations (e.g., deep packet inspection) and accounting purposes (e.g., resource utilization accounting) [12]. The key advantage of this technique, compared with active counterparts, is its non-intrusive nature. The active monitoring schema generates an unavoidable overhead due to the probes launched for testing the system, which if not managed properly, will cause a harmful impact

to the target system. On the other hand, passive data collection observes the whole system in a silent way, without bringing any network overhead, and is able to provide fine-grained system information. However, the passive data collection deploys heavy instrumentation on the target system. Once deployed it is not easy to be modified at runtime, making it unfeasible to meet a changing monitoring requirement in an on-the-fly manner. Moreover, in order to manage the monitoring procedure at a relative low cost, it is not advisable to exert much complex computation work on the large amount of fine-grained data at runtime. As a consequence, passively collected data are more often used in off-line analysis, such as system log inspection [179] and root causes diagnosis [172].

In contrary, the active data collection relies on the capability of sending test probes to the target system, tracing them and measuring obtained results. According to [136] a probe is a test transaction whose outcome depends on some of the system components. The *ping* and *trace route* commands are two widely used probes in distributed systems for detecting network availability. Other specific probes, such as glite [98] command *srm-ls* for testing the list ability from a computing element (CE) to a storage element (SE) in the grid environment, provide more sophisticated, application-level functionalities. A typical usage of the active monitoring in a distributed system is the end-to-end performance estimation between two nodes, such as round-trip time of HTTP requests, connection bandwidth or availability, network delays or latencies. Applications of such end-to-end performance estimation include selecting the lowest-latency peers to communicate with in P2P networks, or choosing a high-bandwidth storage element (SE) from/towards which to download/upload files in grid system.

Extensive applications can be categorized into the *active* class. [5] proposes QMON, a QoS-capable monitoring system which adapts its monitoring frequency and data volumes to obtain a balance between monitoring overheads and the improvements in utility of the performance data. Nagios [2] is an open source tool for monitoring IT infrastructure, which provides monitoring of hosts and services in both active and passive ways. The active monitoring is initiated by the Nagios process and run on a predefined schedule, while the passive monitoring is initiated and performed by external applications, whose results are submitted to Nagios for processing. [111] proposed another hybrid topology-aware approach for monitoring the grid network. Passive measurements are first used to obtain topology information in the aim of discovering bottleneck links. These bottleneck links are later

monitored actively. In order to manage the invasive measurements at a low level, active measurements are only used when none application is running, and passive measurements are applied when there are running applications. ActiveMon [1] is monitoring framework for generating and analysis of active measurements which can be used for routine network health monitoring. Network metrics such as route changes, delay, jitter, loss, bandwidth and MOS are supported by ActiveMon and are initiated in a regulated and non-conflicting fashion between multiple measurement servers distributed at strategic points in a network. It can also be used to aid in determining end-to-end network performance bottlenecks along measured paths.

The active approach generates extra artificial traffic or requests to the system in a controllable way, which is the probe schedule. This explicit control opens the possibility of a dynamic and adaptive probe dispatch. Because probes are pure overhead, the general objective is obviously an optimal tradeoff between the number of probes, and the quality of their description of faults. Thus, two issues have to be addressed. The first one is the goal: what description are we looking for? The second one is the methods to reach this goal: are they amenable to adaptivity, and most importantly, some level of anytime-ness, in our case being able to return meaningful information even with a suboptimal number of probes.

2.2.2 Fault analysis

Given the data collected either passively or actively, fault analysis plays a key role in fault management as it reveals valuable information of the target system. Generally, fault analysis can be seen as a task of detecting, diagnosing or inferencing the malfunction or malicious behaviors of a given system or its involved components. Two types of causes may be responsible for faults in distributed systems. Software errors which usually existed in the system design, implementation and configurations can result in a functional fault or performance degeneration of the system. On the other hand, malicious nodes in a system such as attackers in an open P2P community may render the whole system disorganized or even chaotic.

Detection and diagnosis

Without loss of generality, a target system can be seen as a set of hardware and software components, then the task of fault analysis is to discover the status of involved components,

either functioning correctly (UP) or not (DOWN). Fault detection is the step of discovering if any of the components is DOWN, while diagnosis aims at exhibiting all DOWN components or root causes. Fault inference focuses on inferring unknown status of some components based on information of the others. More exactly, in detection [180], information about the execution of a distributed system are verified against some predefined specifications, which can be in the form of expected properties, state machine model, or reference implementation. Once any fault is detected, fault diagnosis is utilized to identify the nature of the fault and track the root causes. Fault inference, on the other hand, does not inspect the details of a fault, but instead looking for a global view of the system based on partial observation of it.

[180] categorized the mechanisms of fault detection into three classes, i.e., invariant checking, reference implementation, and model checking. In the invariant checking, the desired system behavior is defined as a set of invariants and information about system state is collected either by inserting additional statement (e.g., Pip [133] and P2Monitor [151]) or by modifying underlying operating system (e.g., WiDs [109] and D3S [108]). The acquired system state data are checked against invariants either by online assertions or through offline analysis. Reference implementation (e.g., PeerReview [68]) assumes a deterministic model of system behaviors, and detects faults through a comparison between the actual system and its corresponding reference implementation. In model checking, system behavior is formalized as a state machine for simulating all possible execution paths. The target system is driven to follow each of the execution paths to check whether it behaves correctly.

Further more, fault diagnosis can be divided into three classes [180]: log-based causality analysis, deterministic replay and statistical inference. The log-based causality analysis uses statements woven in the source code of target system to expose expected system state. Based on the log traces of system state, it is able to reason about the causality paths and reveal the root causes of faults. Pip [133], XTrace [52] and D3Scite [108] are representative examples of this approach. In deterministic replay, all non-deterministic events are recorded at runtime through a set of predefined watchpoints and breakpoints. Once a system fault is detected, these collected events are used for reproducing as well as diagnosing the fault. Liblog [59], Friday [58], WiDS [109], MaceMC [89] can be categorized into this class. In statistical inference, a target system is modeled as a set of system components, which are treated as black-boxes. Run time system state are recorded in the form of execution paths consisting of involved system components. Algorithms such as clustering are used to reveal

the correlation of detected faults and recorded components. Magpie [18] and Pinpoint [34] are systems adopt this mechanism.

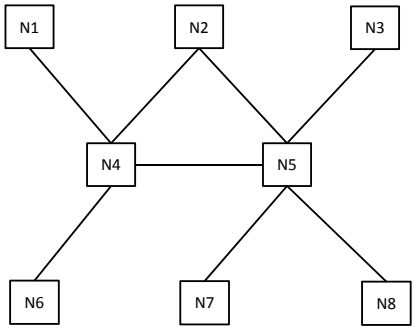
All these approaches mentioned above depend on the passive data collection. For example, both the invariant checking and log-based causality analysis rely on the collection of logs and snapshots either by annotating source code or injecting state exposers into the underlying operating system. In reference implementation, non-deterministic events, such as read/write of files, are recorded at runtime. Deterministic replay uses watchpoints or breakpoints to monitor system events corresponding to data or control flow. Statistical inference also record the involved system components on an execution path at runtime.

On the other hand, methods for diagnosing the actively collected data usually assume a dependency model between the probes and the system components. A diagnostic approach based on end-to-end probing was firstly proposed in [56] and later developed in [30] and [31]. Suppose a distributed system consists of a set of nodes $N = \{N_1, N_2, ..., N_n\}$. Each node represents a monitored system component in the system, e.g., a server, a database, a router and etc., and can have the state of either **up** (function correctly) or **down** (otherwise). A probe p, which tests a subset of N, can either have the result OK, if all its tested components are **up**, or failed, if any of its tested components is **down**. We further use $N(p)$ to denote the set of components tested by p. Given a probe set $P = \{P_1, P_2, ..., P_k\}$ and node set $N = \{N_1, N_2, ..., N_n\}$, the dependency matrix $D_{P,N}$ is defined by[136]:

$$\begin{aligned} D_{P,N}(i,j) &= 1 \quad if N_j \cap N(p_i) \neq \phi \\ &= 0 \quad otherwise. \end{aligned}$$

where each row of $D_{P,N}$ represents a probe and each column represents a node. Figure 2-2 illustrates the core idea of this approach. Probe p_{16}, as shown in the figure, tests the path from N_1 to N_6. Its components set $N(p_{16}) = \{N_1, N_4, N_6\}$, thus the value of row p_{16} is $(1, 0, 0, 1, 0, 1, 0, 0)$.

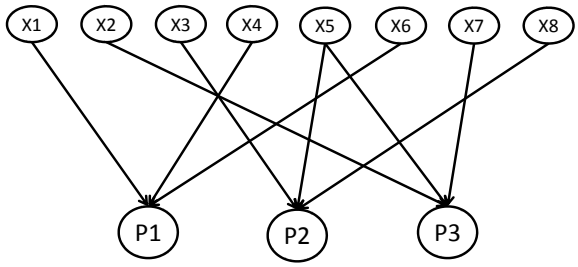
Besides the dependency matrix, Bayesian network is another commonly used model for fault diagnosis [137] [136] [36] [177] [22]. The Bayesian network is a graph model which depicts the relationship between network nodes and probes (The network nodes and probes share the same definition as in that in the dependency matrix). For example, the dependency matrix in Fig. 2-2(b) can be converted into a two-layer Bayesian network as in Fig. 2-2(c).



(a) End-to-end

	N1	N2	N3	N4	N5	N6	N7	N8
Probe1=p_{16}	1	0	0	1	0	1	0	0
Probe2=p_{38}	0	0	1	0	1	0	0	1
Probe3=p_{27}	0	1	0	0	1	0	1	0

(b) Dependency matrix



(c) Bayesian network

Figure 2-2: Illustrative system, dependency matrix and Bayesian network

Every vertex in the figure represents a random variable, the upper layer variables correspond to system components while the lower-layer denotes variables of probes. The set of parents of p_j (set of components tested by probe p_j) is denoted as $N(p_j)$. Joint distribution of this Bayesian network can be written as follows [137]:

$$Pr(x,p) = \prod_{i=1}^{n} Pr(x_i) \prod_{j=1}^{n} Pr(p_j|N(p_j)), \tag{2.1}$$

assuming that state variables x_i are marginally independent, and that each probe result depends only on components covered by it. $Pr(x_i)$ specifies the prior probability of system state, while the conditional probability $Pr(p_j|N(p_j))$ depicts the dependency of probe result on the components tested. Diagnosis of fault is accomplished by probabilistic inference using Bayesian network, i.e., finding $Pr(Z|Y=y)$, the posterior probability of set of variables Z given observations of some other variables $Pr(Y=y)$. For example, we could update the fault probability of every single node, based on the tested probe results.

As described in Fig. 2-3, there are several common steps to carry out to complete the fault diagnosis of a distributed system using active probing. At the beginning, an initial set of probes are selected either randomly or heuristically, then probes are queried to the running system and the returned results are analyzed. If any other information is needed for the identification of a detected problem, probes are further selected, queried and analyzed until the problem is diagnosed.

In the active data collection context, the task of fault detection is to find the smallest set of probes such that, whenever a problem occurs, some probes will fail, whereas task of fault diagnosis is to find the smallest probes set such that, once a problem has occurred, the exact problem can be identified from the probes' results. Both cases assume a priori knowledge of the components of the system, as well as knowledge of the dependency matrix, which describes the outcome of each probe given the status (*up* or *down*) of these components. However, as shown in [136], finding the optimal set for both problems are NP-hard.

Discussion

The obvious advantage of detection and diagnosis is that they provide an accurate explanation of the failure, by exhibiting culprits. On the other hand, it strongly relies on a priori knowledge which components are required for a probe to succeed through the dependency

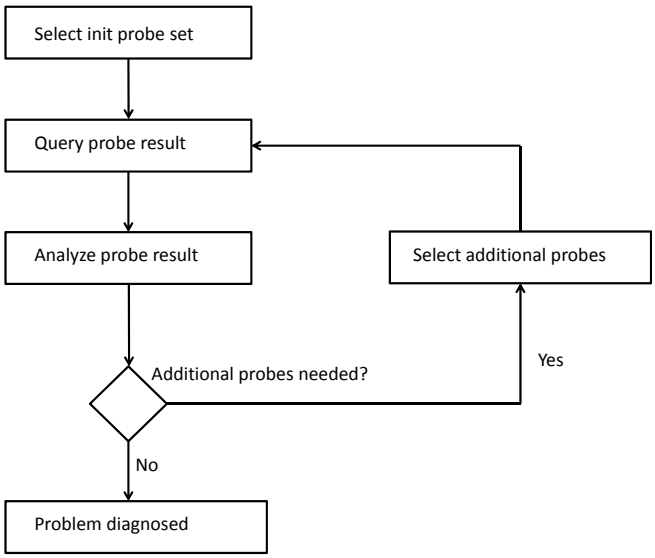


Figure 2-3: Diagnosis flow chart

matrix. However, being accurate implies more complexity. One thing brings complexity in active monitoring is the deterministic dependency model which gives an explicit description of the relationship between probes and components tested. The basic assumption of deterministic dependency can be questioned along multiple axes in the case of large scale dynamic systems.

Firstly, is it simply possible to have a complete a priori knowledge of the structure of the system? These systems are themselves very complex: they integrate heterogeneous hardware and distributed software stacks which functionalities are only partially known. Deterministic dependency analysis requires an accurate graph of the involved components and their relationships in a system with tens of thousands of components, as well as an exact knowledge of the relationship of a probe and a component. For massively distributed systems, Lamport's famous definition "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable" applies very concretely, and Section 2.2.4 will show an example.

Secondly, the deterministic dependency model assumes that if one of the components being tested is failed, then all tests cover this component will fail [136]. However, large scale systems are intrinsic dynamic and evolving. In other words, not only is the structure difficult to describe, but the very concept of a static structure might inadequately model

these systems because of system dynamics.

Dynamism of a distributed system can be introduced in several ways:

High availability In order to provide high availability of a distributed system, system resources like processes and data are always replicated. Moreover, replication, concurrency as well as failures should be transparent to end users so as to equip a smooth and fluent user experience. Once any failure exhibited in these replications, a rearrange of the relationship will carried out among involved components, thus dynamism appears [57].

Elasticity and scalability In the cloud computing paradigm, servers are required to adapt rapidly to dynamic workload such as the frequently changing amount of client requests per time unit. As a consequence, the need of elasticity and scalability are introduced to manage variable and potentially high workloads, which intensifies the inherent dynamism of such systems and applications [176].

Dynamic environment The environments where distributed systems operated are complex and dynamic [57]. Dynamic environments like changing interconnection between components, power supply levels, CPU/network bandwidth, latency/jitter, unpredictable user behaviors have a deep interaction with the running system, and without any omen, will affect or even crash the system.

Thirdly, multiple simultaneous faults are common in large scale distributed systems. Unfortunately, when diagnosing multiple simultaneous faults the complexity of deterministic model either using dependency matrix or Bayesian network increases significantly with the number of possible faults. More exactly, to diagnose multiple simultaneous faults the number of probes needed by dependency matrix increases exponentially in the number of possible faults. Then again, the complexity of Bayesian network increases with an increasing number of simultaneous faults and depends on the efficiency of representing a priori knowledge $Pr(X)$ as well as the efficiency of probabilistic inference and the computation cost of active probe selection [137].

To sum up, for massively distributed systems, assuming a priori knowledge is hazardous in principle and hard to achieve in practice. Therefore, in this part we only deal with the end-to-end fault inference problem with the goal of minimizing the number of launched probes.

Fault inference

Fault inference, instead of exhibiting all possible faulty components, focuses on inferring the status of system nodes based on observed results of other nodes. Various applications like congestion control [166] [147], streaming application [150], QoS verification [5], download server selection [97] and service selection [171] can benefit from fault inference. In this case, the overall infrastructure is a black box, with no a priori knowledge of its structure. End-to-end probes are designed to test a functional property of this black-box, and can be meaningfully replicated in the system. For instance, in the example that will be further described in Section 2.2.3, the functionality is related to file access, and the probes are launched from the computing nodes to the storage nodes. Then, we formulate the objective as a classification problem: from a selection of the probes (the training set), the outcomes of the other probes are inferred.

In the following, this inference action will be called fault prediction, because the term is traditional for the relevant contexts in Machine learning: supervised learning at large, and more specifically Collaborative Prediction. However, it must be stressed that there is no temporal dimension here: the goal is to build a generative model of the probe outcome. In practice, of course, this generative model will be used as a predictive one.

2.2.3 Motivating application

The European Grid Initiative

Two approaches have been proposed to provide computational resources at a large scale: the grid and the cloud. In the grid model, institutions acquire resources and make them available to e-science users; the key point is sharing, as stated by Foster et al. in [54]: *"resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs."* In the cloud model, the resources are leased to users, and the key point is the capacity of dynamic resource provisioning (on-demand availability), coined as *elasticity* by Amazon EC2 [9]. Since 2000, a production grid at European level and beyond has been developed through the successive DataGrid (2001-2004), EGEE (2004-2010) and EGI-Inspire (2010-2014) projects co-funded by the EU. It enables access to computing resources for European researchers from all fields of science, including high energy physics, humanities, biology and more. In 2013,

the infrastructure federates some 350 sites world-wide, gathering more than 250,000 cores, which makes it the largest non-profit distributed system in the world.

Organized sharing is a fundamental requirement for large scientific collaborations running immensely large simulations on a timescale of tens of years, such as in the High Energy Physics (HEP) community. This characteristic has had considerable impact on the grid design, with the concept of *Virtual Organizations* (VO): the access rights to EGI are primarily organized along this concept. The resulting complexity is high: the general logic of IT systems is more oriented towards individual than collective rights, and the VOs are orthogonal to institutional ownership and administration of the resources. The EGI initiative, which steers the EGI-Inspire infrastructure, is presently experimenting a Federated Cloud infrastructure; the main difference is that the local node is no more a batch system, but becomes a cloud; while this has significant effects on the Quality of Experience for the final end user (elasticity and responsiveness), the fundamental issue of sharing remains, with the associated added complexity.

Hardware and software failures are intrinsic to such large-scale systems. Resource availability in production is about 90%, and middleware e.g., gLite [98] , Globus [53] or ARC [46] cannot handle this without substantial human intervention. Each of the 200 VOs has to be specifically configured on its supporting sites, which adds complexity and introduces extra failures. User communities exploit two strategies to cope with faults: overlay middleware e.g., DIRAC [160], DIANE [120], AliEn [17] and PaNDA [114], implements specific fault-tolerance strategies to isolate users from the vagaries of the infrastructure; and monitoring identifies problems and quantifies performance w.r.t. quality of service agreements.

Monitoring for the Biomed VO

The target system of this work is the Biomed VO. Biomed has access to 243 Computing Elements (CEs) and 131 Storage Elements (SEs), approximately. CEs are shares of computing resources, implemented as queues of each site manager (e.g., PBS), and SEs are shares of storage resources; the formal definition is part of the Glue Information model [11]. Testing the availability of all CE-SE pairs is one of the most challenging issues encountered daily by monitoring operators. The original method is brute force: it periodically launches a fully distributed all-pairs availability test, for a total of 31833 tests, multiplied by the number of capacities to test at each run. Human operators cannot handle so many results; in practice,

only a few issues are reported, with questionable selection criteria.

A significant reduction of the number of tests providing nearly similar availability evaluation performance would be highly beneficial: besides a better frequency/intrusiveness trade-off, the selection of reported incidents would become more informative.

2.2.4 Diagnosis at work

Figure 2-4 illustrates a typical view of software and hardware stack for a Grid system [15]. Four layers are considered here: fabric, core middleware, user-level middleware, and applications and portals layers. The fabric layer is the fundamental of a running grid, which consists of distributed resources such as computers, networks, disks and scientific instruments. The core grid middleware provides many key services like process management, co-allocation of resources, storage access, security, and etc. It can be seen as an abstraction of the fabric layer for providing a consistent view to access distributed resources. Higher level of abstractions are further provided by the user level middleware in the form of application development environments, resource brokers and task schedulers. The top layer is the Grid application and portals, where end-users can use it for submitting and collecting their jobs on remote resources through the Web-enabled services.

Given on the layers of the grid architecture, we designed a diagnosis system for testing the availability of the CEs and SEs in the Biomed VO. For simplicity, only the key services like *srmls*, *lcgcp*, *lcgcr*, *LFC* (logical file catalog), *VOMS* (Virtual Organization Manage System), *BDII* (Berkeley Database Information Index), *SRM* (Storage Resource Manager), network and a few corresponding hardwares are considered. Figure 2-5 gives a basic idea of the relationship between designed probes and components. One thing to mention is that the variable in the brackets after each component denotes the number of that component. The definition of the variables is given as following:

- G = 1, number of user account, can easily be higher if we want to test several different types of user account;
- k = 131, number of SEs;
- n = 243, number of CEs, i.e., probe station;
- N = 1, number of gridftp server, currently we are not able to identify the gridftp servers behind an SRM endpoint, whose actual number is several hundreds.

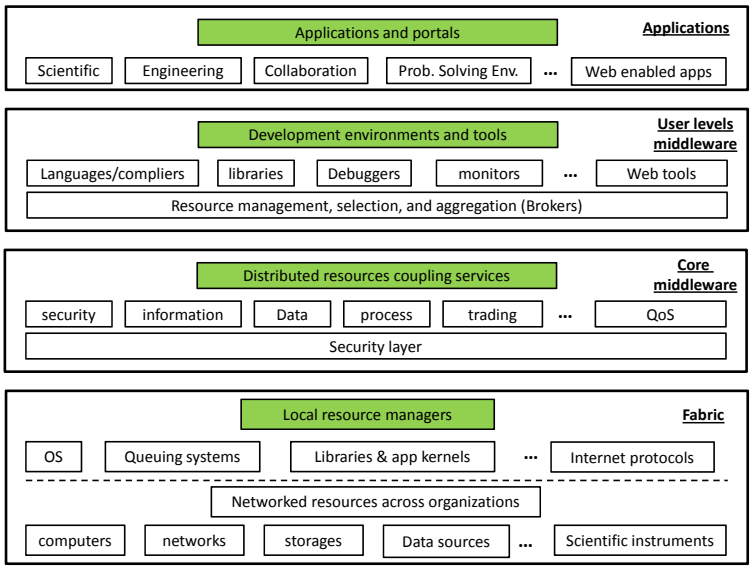


Figure 2-4: Grid architecture

Let DM represents the dependency matrix, number of rows in DM is: $n * (4 + k * 5) = 160,137$, and number of columns in DM is: $4*n+3*k+9 = 1374$. thus the total number of entries in DM is: 220,028,238. Even though, this huge matrix is only a coarse approximation of the system, without considering many components like libraries, queuing system, brokers and etc. The complex dependency model hinders us from a deeper exploration on the diagnosis way, and making us resort to the fault inference solution.

	Sm-ls cmd	lfc cmd	lcg cmd	LFC service	LFC hd	N1	N2 (k)	SRM (k)	SRM hd (k)	N3	VOMS	VOMS hd	N4 (n)	BDII config (n)	BDII (n)	BDII hd (n)
Lcg-cr (G*k*n*N)	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Lfc-ls (G * n)	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	0
nmap (LFC is closed)	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
nmap (LFC is open) (n)	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0
Voms-proxy-init (only on UI)	0	0	0	0	0	0	0	0	0	1	1	1	1,only once	0	0	0
nmap (VOMS is open)	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0
Srm-ls (k * n * G)	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0
nmap (SRM is open) (k * n)	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0
Lcg-infosites (local BDII) (n)	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1
nmap BDII (n)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Lcg-cp (G * k * n)	0	0	1	1	1	1	1	1	1	0	0	0	1	1	1	1

Figure 2-5: Dependency matrix example

Chapter 3

Collaborative Fault Monitoring

This chapter addresses the knowledge building step, and the knowledge/monitoring loop, with the formulation of the fault inference task as a Collaborative Prediction (CP) problem. Our main goal is to highlight in which directions the general CP framework should be adapted to this new application area. Results on an extensive experimental dataset from the EGI grid demonstrate the excellent performance of a combination of the Maximum Margin Matrix Factorization approach to CP and Active Learning.

3.1 Collaborative prediction for fault inference

In this section we formalize the problem of fault inference in a distributed system as a collaborative prediction task.

3.1.1 Motivation

We first show that, from a strictly formal point of view, end-to-end performance monitoring such as fault inference can be naturally modeled as a matrix completion problem. Collaborative Prediction (CP) associated with end-to-end probing, with the components structure considered as a black box, participates in the general Quality of Experience (QoE) approach [134]. More precisely, an important ingredient separating QoE from QoS is binary (possibly extended to discrete) classification. Most work in this area is devoted to network-based services (e.g., among many others [159]). Z. Zheng and M.R. Lyu propose explicit users collaboration for estimating failure probabilities [178]; while their end-to-end framework is related to ours, the goal is more in line with QoS (estimating the full distribution of

probability instead of binary classification), and the correlation amongst users or services is modeled by ad hoc methods.

The key insight to go beyond such ad-hoc methods is to leverage the considerable amount of theoretical and experimental work developed for Collaborative Prediction. Before QoE became a popular keyword, Rish and Tesauro [138] explored the combination of CP and Active probing for the selection of good servers in various distributed and P2P systems. Our work combines the goal of proposing fault-free services to the user exemplified in [136, 178], and the CP approach of [138]. To actually solve the problem, Collaborative prediction, which is often associated with matrix completion, is a good candidate; however, its efficiency strongly depends on the selected algorithm. In the next section, we briefly present collaborative prediction, and motivate its relevance by the empirical properties of the fault production process in large scale distributed systems.

3.1.2 Fault inference as a matrix completion problem

End-to-end performance estimation has many applications like congestion control [166] [147], streaming application [150], QoS verification [5], download server selection [97] and service selection [171]. It can be converted into a matrix completion problem naturally. Take the download server selection for example, in the content distribution networks (CDNs) replications of content servers are geographically distributed across the Internet and clients are redirected to the server with the lowest latency[97], such that the download time of a content from the server is reduced. In order to measure the latency, active probes like *ping* and *traceroute* are launched at runtime to gather network information between clients and servers. The collected measurements form a latency matrix between clients and servers. However due to the huge number of clients on the Internet, it is too costly to exhaustively measure each client-server pair. Thus it is more appropriate to only measure part of the matrix entries and estimate the left others. This formulation is fundamentally a matrix completion task: given a very sparse client-by-server latency matrix, whose non-zero entries represent known latencies, predict the unknown entries of the matrix[138].

End-to-end fault inference has strong relation with the server selection issue: the performance that we want to estimate is the capacity to deliver the requested service. Then, end-to-end fault inference can be formalized as a matrix completion problem as well, where each entry in the matrix will represent the end-to-end service functionality status between cor-

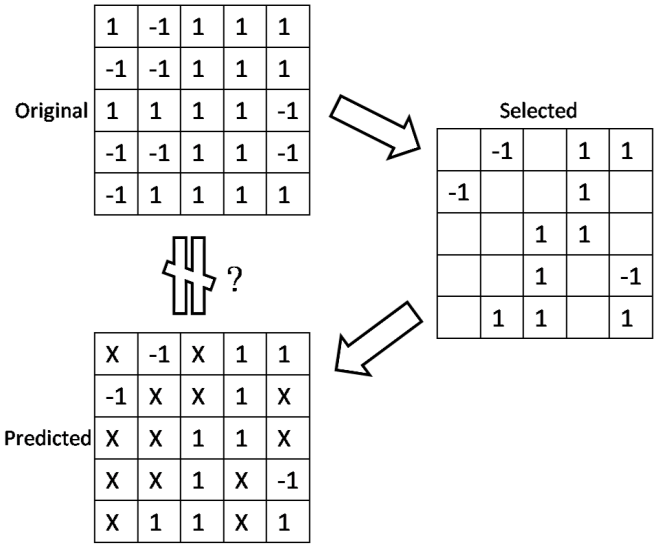


Figure 3-1: Illustration of matrix recovery

responding service provider and requester. From the fault inference as a matrix completion task, we focus on fault inference between end-to-end points, and the overall infrastructure is treated as a black box, with no a priori knowledge of its structure.

For most of this chapter, a basic assumption is *staticity*. By this, we mean that the underlying physical process that generates faults does not change over the time window where the input data are collected and the prediction is made. Then, it is natural to formalize the inference task as a *matrix completion* problem: if X is the observed (sparse) matrix, the CP problem is to find a full, real-valued, matrix Y of the same size that approximates X, i.e., that minimizes some measures of discrepancy between Y and X. When Y is required to be equal to X on the known entries, the problem is termed *exact completion*, and *approximate* otherwise.

For the fault inference problem, X is the results of the selected probes and Y is the predicted matrix. An illustration of *selected* and *predicted* matrices are given in Fig. 3-1. For now, the unknown matrix Y will be assumed to be deterministic; then, the data available for prediction can be considered as a sample of the entries of the full deterministic unknown matrix. The probe selection process defines the sampling distribution.

With such a general setting, the problem is hopelessly ill-defined: in order to guess the missing entries in Y, *some* assumptions have to be made about the matrix to recover Y. In recent period, the common assumption has been to look for *low-rank* matrices. The motivation for the low-rank objective function stems from the fact that the underlying

structure of the matrices in many applications can be formalized as a matrix completion problem.

For simplicity, at this level of generality, we state the problem in the exact completion form; the actual measure of discrepancy will be introduced later. Let S be the set of known entries; Y is the solution of

$$\begin{aligned} &\text{Minimize } \operatorname{rank}(Y) \\ &\text{subject to } Y_{ij} = X_{ij} \text{ for all } (i,j) \in S \end{aligned} \tag{3.1}$$

The existence and unicity of a solution to 3.1 is a complicated problem (see e.g., [32]). Anyway, this formulation is not very helpful, as rank minimization is NP-hard and not feasible practically even for small sizes. However, it has paved the way for efficient algorithms, both for exact and approximate completion. We defer the presentation of these algorithms to Section 3.1.5. Before, we have to examine the relevance of the low rank hypothesis in our case.

3.1.3 Collaborative prediction

CP and matrix completion

CP is originally a technique for predicting unknown ratings of products for a particular user, based on observed data from other users and products. It can be applied to various domains such as online recommendation, link prediction and end-to-end performance estimation and etc.. One important characteristics of CP is that it requests no additional information for accomplishing the prediction task. Take the user-movie recommendation for example, the only input for CP is a partially observed rating matrix whose observed entries are the ratings given by users on those movies they have already watched. In contrary, conventional feature-based prediction approaches (e.g., content based prediction) request external information of user features (e.g., gender, age) and movie features (e.g., genre, year, actors, external reviews) to fill the partially observed matrix. Since the rating information is collaboratively shared among users in the CP, better performance of using CP than of using feature-based methods is observed in a sparse matrix.

A common approach to CP is to fit a low-dimensional factor model to the data, i.e., learn a hidden structure for describing the data. The foundation for the success of a

low-dimensional factor model relies on the hypothesis that a small number of hidden and partially shared factors affect the matrix entries. For the end-to-end fault inference case, the latent factors learned by CP can be the common shared components or services between nodes. For example, two nodes in the system may share factors like geo-location (with associated network connectivity issues) or use of a particular instance of any middleware service (e.g., brokering, authentication), such that the functionality between nodes may be affected similarly.

In a *linear factor* model, each factor can be seen as a preference vector. Take the user-movie case for example, each user's preference is represented by a linear combination of these factors with user-specific coefficients (similarly for the movies). Thus, for n users and m movies, the preferences of a k factor model can be expressed by a product of a $n \times k$ coefficient matrix U and a $k \times m$ factor matrix V'. This factor model describes each user through a coefficient vector, capturing each user's underlying preference separately.

This amounts to express the Y matrix of (3.1) as

$$Y = UV', \tag{3.2}$$

thus looking for a low-rank matrix is equivalent to looking for a low-dimensional factorization. This approach is termed *matrix factorization*.

If one of the matrices, say U, is fixed, and only V needs to be learned, then fitting each column of the target matrix X is a separate linear prediction problem. Each row of U is a feature vector, and each column of V' is a linear predictor, predicting the entries in the corresponding column of X based on the features in U. However, in our case, both U and V are unknown and must be learned. This can be considered as equivalent to extract feature vectors (rows in U) for each of the rows of X, in order to create good linear predictors (columns of V') across all of the prediction problems (columns of X) concurrently. The features are learned without any external information or constraints. This would be impossible for a single prediction task. The enabling characteristic of collaborative prediction is that the prediction tasks are related, and the same features can be used for all of them, though with different importances.

For completeness, we recall the definition of singular value decomposition (SVD). SVD [62] is a factorization of a real or complex matrix. For a full matrix Y, a compact SVD repre-

sentation of Y is given by

$$Y = U\Sigma V^T, \tag{3.3}$$

if we use R to denote the rank of Y, then U and V are orthogonal matrices of sizes $M \times R$ and $N \times R$, respectively, and Σ is a diagonal matrix of singular values $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_R$. A common approach in low-rank matrix factorization is to fit a target matrix X with a low rank matrix Y through SVD and by minimizing the sum-squared loss. This method, as we show in later section, is not well applicable in the collaborative prediction case where data classes in X are highly imbalanced.

Once again, before going to our main point, we examine the alternative approaches to matrix factorization that have been proposed in the context of Collaborative Prediction.

CP methods

[154] categorized various collaborative prediction techniques into three classes, i.e., memory based CP, model based CP and hybrid recommender. Neighborhood methods like KNN (K-nearest neighbors) and item or user based top-N methods are representative examples of memory based CP. In memory based CP, the entire or a sample of the user-item dataset is used to produce a prediction. Every user contributes to a group of people with similar interests, where each user's preference is generated by identifying its neighbors in the group. Normally, memory based CP enjoys its advantages in aspects like easy implementation, incremental new data processing, need no consideration of item's content and scale well with co-rated items. However, on the downside, its disadvantages are also obvious: dependent on human ratings, poor performance on sparse data, can not recommend new users or items and limited scalability for large datasets. In our case, the incremental capacity is irrelevant, and the poor performance on sparse data is unacceptable.

Model based approaches are the state of the art in Collaborative Prediction. They include Bayesian models, clustering models, and dependency networks; they learn a complex pattern from the training data, and then make predictions on the test data or real-world data, based on the learned models. Both categorical and numerical data can be easily handled by different model methods. Compared to memory based CP, model based CP has been shown to provide better performance on addressing problems like data sparsity, scalability and etc. Moreover, an improved prediction performance as well as an intuitive

interpretability are also yielded by applying the model based methods. Nonetheless, to build a useful model, especially on a large dataset, it is computationally expensive. Further, the trade-off between prediction performance and scalability needs careful tuning.

The matrix factorization approach [152, 145, 156] can be considered as a special and elementary form of the model-based approach: the generic model is linear, and its coefficients are learned from the data. The *Maximum Margin Matrix Factorization (MMMF)* [152] will be described in more details later.

In general, matrix factorization considers that all data are equally relevant to the prediction task: the values are generated using the same factor vector. This approach, which assumes a strong homogeneity of the data, has been termed *continuous latent factor* by [113]. Other *continuous latent factor* methods such as PMF [118] and BPMF[144] can be seen as graphical models in which entries in X are directly connected with latent factor variables in a probabilistic graph. [144] is oriented towards imbalanced ratings (some users rate a lot, other not). This leads to two independent categories of hidden information: the topics, and their distribution, instead of only one, the topics, which is not at all our case, as we can freely select the probes.

Despite good prediction performance, these latent factor methods are blamed for their static way of predicting missing entries in X, as all the missing entries in a row of X are produced by using a same factor vector, without considering the context [113]. Therefore, various improvements have been carried out to make the continuous latent factor model context-aware. This alternative model-based approach considers that the data should be contextualized through *bias* terms included in the model. The most frequent justification for such contextualization is temporal variation; in recommendation systems, the context can be the mood of the user, or even the fact that two users share a common Internet access.

Y. Koren and the BellKor team, winners of the highly published Netflix prize, proposed explicit modeling of various contexts, including temporal drifts and spikes [94]. They remarked that a basic model, which captures context effects but disregards user-item interactions, explains more of the data variability than the commercial Netflix Cinematch recommender system. However, the model selection for bias is largely based on the specificities of users behavior in movie recommendation, thus not easily extendable. Alternatively, *block models* [126, 7] propose a fully generative model and leverage the older neighborhood method [71] based on clustering. The specific application to CP is Bi-LDA [126]. Al-

though LDA (Latent Dirichlet Allocation) also embodies the discovery of latent (hidden) factors, the key difference between Bi-LDA and matrix factorization is that the relationship (user/product in recommendation systems) is allowed to select a new topic (factor) for each interaction.

In dynamic distributed system, transient failures are well attested, and the frequency of switching between functioning/malfunctioning can be high; such context bias could be anything like a middleware installed temporary down, or too many concurrent write requests issued to a server, making context-awareness appealing. However, Bi-LDA is known to exhibit relatively poor predictive performance, probably due to an exclusive modeling of interaction of clusters (through topics); in other words, the expression of the specificity of individual interactions (this particular user/CE with this particular movie/SE) is lost.

Recently, Mackey proposed a Bayesian approach to reconcile Matrix Factorization and probabilistic topic selection with Mixed Membership Matrix Factorization (M^3F) [113], introducing context dependence in a more general way than the a priori formulation of [93]. Moreover, in recommendation systems, the greatest performance improvements with M^3F occur for the high-variance, sparsely-rated objects, suggesting a good capacity at capturing the transients that are a serious issue for system operation managers.

Specificities of CP for fault inference

The key point is that monitoring large scale distributed systems differs from usual CP applications (personalized recommendation), in two major ways. On the bright side, while users cannot be queried for specific recommendations, probes can be launched at will. On the downside, the distribution of the probe results is highly skewed, faults being a small fraction of the total population. In turn, the unbalanced distributions stem from two origins: firstly, fault causes are hopefully rare; and second, some of the faults are transients. In the recent years, CP methods highlighting the role of various bias have received a lot of attention, partially due to their success in the BellKor solution that won the Netflix challenge [95], and specifically address time variability.

By using CP in end-to-end fault inference, we basically assume an idealized situation where the acquired data are a snapshot of a stable one, however, under a continuous monitoring situation, where faults are changing from one time window to another, a sequential analysis method is needed. We present our work on sequential collaborative fault inference

in Chapter 4.

3.1.4 Goals and methods

Objectives

As described in the previous section, the goal is to minimize the number of probes, for a given quality of prediction. The precise definition of quality is deferred to 3.2.2, but is naturally related to the mis-prediction risks. Minimizing the number of probes encompasses two distinct issues: intelligent probe selection, and an efficient matrix completion algorithm.

Probe selection strategies

Algorithm 1: Generic active probing algorithm

input : Initial partially observed binary(-1/+1) matrix M_0, threshold λ, max # of new samples N, active-sampling heuristic h
output : Full binary-valued matrix M^{T_i} predicting unobserved entries of M_0
initialize: Initialize the vars

$S(T_0) = S(M_0)$ /*currently observed entries set*/ ;
$i = 0$ /*current iteration times*/ ;
$n = 0$ /*current number of new samples*/ ;
while $(n < N)$ **do**
    $M^{T_i} = StandardMC(S(T_i))$ /*Prediction based on observed entries via standard matrix completion (MC) procedure*/ ;
    $S'(T_i) = ActiveSampling(M^{T_i}, h, \lambda)$ /*Actively choose the next set of new samples and query their labels*/ ;
    $S(T_{i+1}) = S(T_i) \cup S'(T_i)$;
    $n = n + \#S'(T_i)$;
    $i = i + 1$;

We consider three probe selection methods.

- **Static-Uniform.** The probes are selected uniformly at random amongst all (CE,SE) pairs. In this setting, the probe selection and the prediction are completely independent: the prediction step has no influence over the choice of the probes. This would be unrealistic in recommendation systems (users do not select uniformly the products they rate amongst all proposed), but can be fully implemented in probe selection. Moreover, for the subsequent prediction task, uniform sampling provides theoretical bounds on the MMMF generalization error.

- **Active Probing.** Active Probing instantiates the general Active Learning approach: incrementally building the learner by querying the information source for labeled examples. With Active Probing, the set of probes is constructed dynamically, with an initial set of probes selected for instance by the Static-Uniform method, and run through the system to get basic information; then, additional probes are selected and launched with the goal of maximizing some measure of information. Algorithm 1 illustrates the process: a predicted matrix is first given by standard Matrix Completion (MC) based on some pre-selected samples (step 5) , then some heuristics are used for filtering the next subset of samples, which are labeled by actually running the probes and observing their outcome (step 6). After several iterations, a final prediction is returned. In this setting, the CP method used for MC impacts the probe selection. In this work, the min-margin heuristic [158] is used for selecting additional probes. Min-margin favors exploration over exploitation: it chooses the probe where the uncertainty of the classification result is maximal, and has been demonstrated to be efficient for CP problems [138].
- **Differentiated costs.** In the two previous methods, the same penalty is associated with both kinds of mispredictions. It might be argued that a false negative (predicting success while the actual result is a failure) is more harmful than a false positive (predicting failure while the actual result is a success), because the federated nature of the computational resources offers multiple options to users. Unbalanced costs (in either direction) arise in many other contexts, e.g., medical testing [119], and can be integrated in the core learning step, as shown in the next section.

3.1.5 Matrix completion

This section first discusses the relevant theoretical results, then presents the Maximum Margin Matrix Factorization algorithm.

Theoretical bounds

Going back to the low-rank formalization of the exact completion problem (eq. 3.1), two questions make sense. Firstly, is there an information theoretic lower bound on the number of entries that should be observed, whatever matrix structure and completion algorithm, in order to recover unequivocally a matrix of given rank? Second, is recovery practically

possible, and at which conditions on the matrix structure? A significant literature has been devoted to this subject (for a review, see [130]), which provides some insight to our results, although our framework is quite different, as we are looking for approximate completion with an unknown rank in a binary context.

Concerning the information theoretic lower bound, [32] shows that it is in the order of $nr \log n$, where n is the linear size of the (square) matrix to recover and its rank is at most r, and the entries are sampled uniformly. For instance, with a 10% sampling and $n = 200$ (we will have use of this case later), the matrix can be recovered uniquely only if its rank is at most 3. A tighter bound with a much simpler proof is presented in [130], and is close to optimality.

A more elementary lower bound is based on the number of degrees of freedom Y through its SVD, and is $2nr - r^2$[32], giving a maximal rank of 10 for our example. This bound is more realistic from an applicative point of view: the information theoretic bound takes into account the coupon collector effect, while the actual sampling (probing) is not bound to be truly uniform, and will avoid repetitions.

An important ingredient to the two first bounds is the role of what this literature calls *coherence*. This indicator can be viewed as an algebraic equivalent of the mutual information between entries. For example, consider the rank one matrix which is equal to 1 in one entry and zeros everywhere else. The coherence is maximum, and all entries must be observed to exactly recover the matrix. More generally, the row (resp. column) coherence describes to which extent each entry can be expected to provide about the same amount of information for its row (resp. column), with minimal coherence when all the entries are equal. In our case, a probe that gives a consistent result on most of its line (or column) is likely to have uncovered a solid cause of failure. The coherence problem will show in the experimental section.

As explained in Section 3.1.3, direct minimization of the rank is not feasible. The main insight for matrix completion is to use instead the trace (or nuclear) norm minimization. Then, Eqn. 3.1 is replaced by:

$$\begin{aligned} &\text{Minimize } ||Y||_{\Sigma} \\ &\text{subject to } Y_{ij} = X_{ij} \text{ for all } (i,j) \in S, \end{aligned} \tag{3.4}$$

where the trace norm $||Y||_\Sigma$ of a matrix Y is defined as the sum of its singular values,

$$||Y||_\Sigma = \sum_{i=1}^{r} \sigma_i(Y).$$

In a nutshell, [131] and [32] show that most low rank matrices could be recovered from most sufficiently large sets of entries by computing the matrix of minimum trace norm that agreed with the provided entries. The minimal size of the set of entries is an increasing function of the coherence indicators. The intuition behind this surrogate objective is that the rank function counts the number of non null singular values, the trace norm sums their amplitude; this is analogous to the fact that the l_1 norm is a useful surrogate for counting the number of non zeros in a vector. Moreover, and very importantly, the trace norm subject to equality constraints can be minimized by convex programming. The technical reason is that the trace norm is the best convex lower bound of the rank function on the set of matrices whose singular values are all bounded by 1[153].

Maximum Margin Matrix Factorization

The Maximum Margin Matrix Factorization (MMMF) [152] exploits the same approach as in exact recovery for approximate recovery. Instead of finding a low rank approximation (e.g., SVD), MMMF minimizes the trace norm of estimated matrix Y under the constraint of no (hard-margin), or small (soft-margin), discrepancy between estimation and observation. This formulation, unlike the low-rank approximation, is convex, thus it is guaranteed to find the global optimal solution. Let S be the set of known entries in X. Two objective functions can be considered.

- Hard-margin: minimize $\|Y\|_\Sigma$ under the constraints

$$Y_{ij}X_{ij} \geq 1 \text{ for all } ij \in S;$$

- Soft-margin: minimize

$$||Y||_\Sigma + C \sum_{ij \in S} \max(0, 1 - Y_{ij}X_{ij}). \tag{3.5}$$

As the minimization procedure produces a real-valued matrix, a decision threshold (e.g., positives values give +1, negatives give -1) gives the final predicted binary matrix.

The soft-margin factorization can be extended with the general robust strategy described by [103] for integrating differentiated costs (or unbalanced positives and negatives examples) in Support Vector Machines: the regularization parameter C in eq. 3.5 is split in two, C_+ (resp. C_-) for positive (resp. negative) examples. The only important parameter is the ratio C_+/C_-.

As we have stated, the key insight in MMMF is to replace the rank minimization with a trace norm ($\|Y\|_\Sigma$) minimization, under the constraint of no or small discrepancy. In this formulation, the dimensionality of the feature space is no longer bounded, but regularized by a low-norm factorization. As a consequence, the only parameter in MMMF is the regularization term between the trace norm and the loss function, which makes it very appealing when compared with methods like graphical models.

Despite the different types of matrix factorization methods introduced in previous sections, we only consider the very simple yet powerful approach, i.e., MMMF, in the first part of the experiment. However, these objective functions, as well as the general matrix completion technique are not the only possible choices for implementing the CP approach to failure prediction. Alternatives which introduces contextual bias, i.e., M^3F, will be discussed in Section 3.5.

3.2 Experimental setting

3.2.1 The data sets

The probes

Different capabilities have to be tested; in the following, we consider three of them: probe *srm-ls* tests the list ability from a CE to a SE, probe *lcg-cr* tests the read ability from a CE to a SE, and probe *lcg-cp* tests the write ability alike. Thus, each CE works as a probe station, launching probes to test the functionalities between itself and each SE. For the Biomed grid a whole set of testing transactions (as we mentioned before: 31833) were launched each day for each of the three probe classes. After nearly two months running, information for 51 validated days were collected. In other words, 51 fully observed SE-by-CE result matrices were obtained for each probe. Figure 3-2 shows the statistical profile

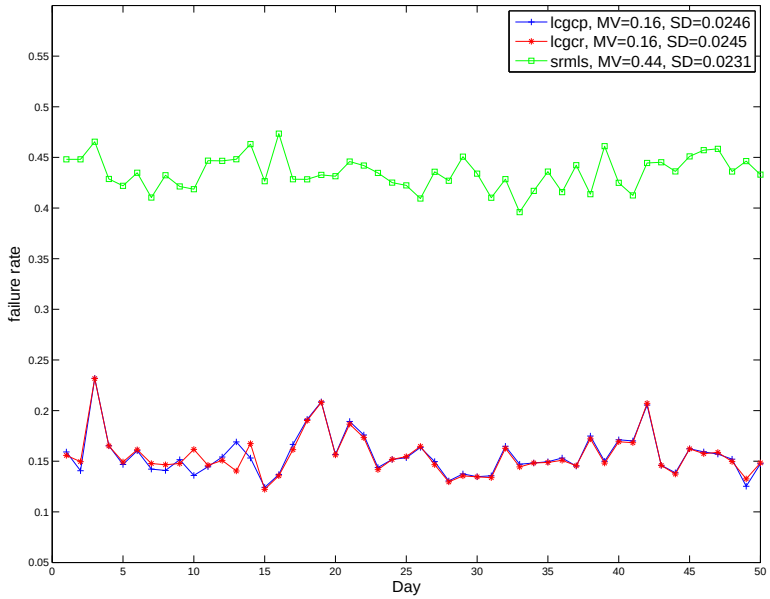


Figure 3-2: Failure rates on a 0.00 - 1.00 scale; the mean and standard deviation are computed over the experiments

of the probe outcomes[1]. Failure rates of *lcg-cp* and *lcg-cr* are almost identical (range from 10% to 25%), while failure of *srm-ls* is significantly higher (ranges from 40% to 50%).

Monitoring issues

The probes themselves are gLite jobs, run by a regular Biomed user. Some of them fail (*rejection*) in the sense that gLite is not able to complete the job, denoting that some job management services may be down or misconfigured (e.g., authentication, brokering etc.). The *rejected probes* entries in Fig. 3-3 shows the ratio of unsuccessful probes over all launched probes in this sense. In the following, we consider only the accepted probes, i.e., those which run to completion, reporting success or failure; this approach amounts to consider that the data access capacities are independent from job management. This is a reasonable hypothesis in a gLite infrastructure because file transfers involved in job management use dedicated storage space independent from the one tested by our probes. Separate testing is good practice in general; in this specific case, the high rejection rate (average 40%) and the high failure rate would act as a massive noise on each other, and

[1]Note that here and on Fig. 3-6 and 3-7(b), only the points associated to each experiment are meaningful; the lines between the experiments are added only for readability purpose.

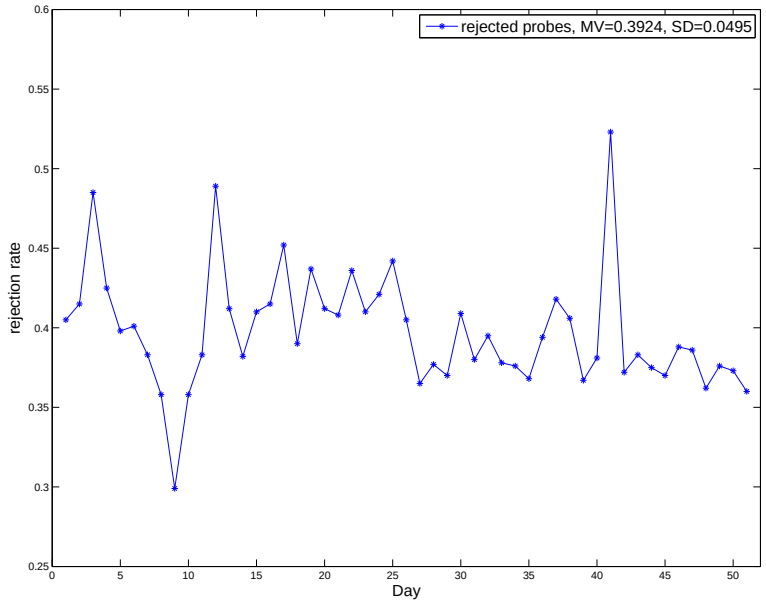


Figure 3-3: Rejection rates on a 0.00 - 1.00 scale; the mean and standard deviation are computed over the experiments

would make CP more difficult if we tried a global approach.

Empirical analysis

We selected five days for detailed performance analysis benchmarks. Table 3.1 shows some basic characteristics of the fully observed entries; the fourth column gives the failure rate when all probes are considered and exemplifies the need for inferring the structure of the apparently massive randomness. Figure 3-4 illustrates the structure for *lcg-cr* and *srm-ls* on day 5 ('07-05-2011'), where rows represent CEs and columns stand for SEs. Each entry is the probe result between the corresponding CE and SE. Black columns correspond to prolonged SE downtimes while black rows are CE failures leading to complete inability to communicate with any SE (e.g., network downtime or configuration issue). These are usually easily detected and reported by human operators with only a few incident reports. The scattered points correspond to local or transient issues, which are very difficult to handle due to the amount of incident reports independently generated. The higher failure rate of *srm-ls* compared with *lcg-cr* appears to be associated with an inadequate port number in some probes, and may be considered as an example of user error.

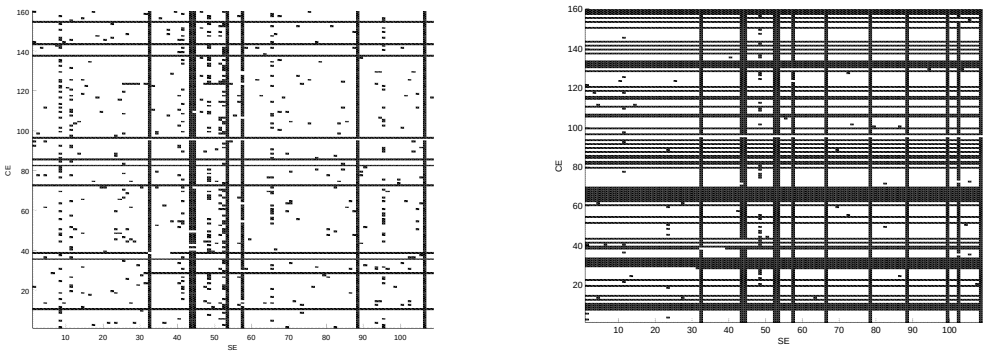
Table 3.1: Five example datasets

Name	Date	Probe	FAILED Native	FAILED Curated
1	2011.04.21	lcg-cp	0.15	0.04
		lcg-cr	0.16	0.05
		srm-ls	0.45	0.02
2	2011.05.14	lcg-cp	0.14	0.03
		lcg-cr	0.15	0.03
		srm-ls	0.43	0.01
3	2011.05.25	lcg-cp	0.16	0.03
		lcg-cr	0.15	0.03
		srm-ls	0.43	0.02
4	2011.06.09	lcg-cp	0.16	0.05
		lcg-cr	0.16	0.05
		srm-ls	0.42	0.01
5	2011.07.05	lcg-cp	0.16	0.06
		lcg-cr	0.16	0.07
		srm-ls	0.45	0.04

The curated dataset

It could be argued that other, global (EGI-wide) monitoring tools should report on these systematic failures, and that the probe selection and prediction methods should be applied only to the more elusive causes of errors. While this is disputable (remember that all probes succeed as jobs, thus at least the CEs are up and running), it is worth assessing the performance of the methods when these systematic errors are eliminated. Therefore, we designed a second set of experiments, with *curated* matrices as the reference fault structure. A curated matrix is a new original matrix, where the lines and columns with only failed entries (black ones in Fig. 3-4) have been removed prior to analysis. Their basic statistics are shown in the last column of Table 3.1. In this case, *srm-ls* shows a lower error rate than the other probes.

Section 3.1.5 has presented informally the theoretical coherence indicator. Getting rid of the systematic faults drastically increases the coherence of the target matrix (recall that coherence as defined in [32] is maximum when only one entry is 0, and minimum when all entries are equal, on a line or on a column), making the theoretical lower bound on the number of samples equivalently larger, and the overall problem more difficult.



(a) lcg-cp (b) srm-ls

Figure 3-4: The CE-SE matrix. Black = FAILED, white= OK

3.2.2 Evaluation methodology

From this dataset, evaluating probe selection is straightforward. Figure 3-1 illustrates the general work-flow of the selection-prediction process. The *Original* matrix is the ground truth: a fully observed result matrix obtained from the all-to-all monitoring runs (Section 2.2.3). Value -1 stands for probe result OK (negative) and 1 means FAILED (positive). The *Selected* matrix is generated by deleting a proper proportion of entries in the *Original* one. In a real-world, probe selection-based, monitoring, the remaining entries would be the only actually launched probes. The *Predicted* matrix is the recovery result generated by the prediction algorithm based on the known entries in *Selected*, where the X entries are now set to 1 or -1. In the real-world scenario they would be delivered to users.

Contrary to the recommendation systems, where there is no ground truth as wish (the users do not rate all products), the collection of data presented in Section 3.2.1 provides the true values. Thus, the classical performance indicators for binary classification can be measured. They describe the various facets of the discrepancy between the *Original* and the *Predicted* matrices.

- Accuracy: the ratio of correctly predicted entries over the total number of entries to be predicted.
- Indicators associated with the risks (confusion matrix): sensitivity, the proportion of actual positives that are correctly predicted; specificity, the proportion of actual negatives that are correctly predicted; precision, the ratio of true positives over all

predicted positives, and the MCC (Matthews Correlation Coefficient), a correlation coefficient between the observed and predicted binary classifications that is relatively insensitive to unbalanced positives and negatives.

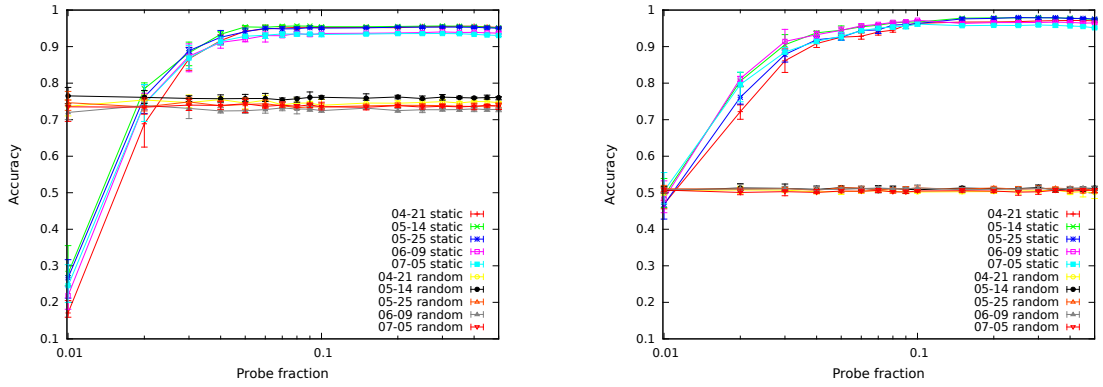
- The Area Under ROC (Receiver Operating Characteristic) Curve (AUC), which summarizes the intrinsic quality of a binary classifier independent of the decision threshold.

The interest of MCC and AUC comes from the fact that, in the optimization step of MMMF, the classification error on the *Selected* matrix is a reasonable estimation of the prediction error, while this hypothesis is less natural for estimating MCC and AUC [84]. Thus, MCC and AUC provide a comparison indicator of the performance of the methods beyond their explicit optimization target.

In order to evaluate the contribution of the prediction (or coupled selection-prediction) methods, we compare their results with a simple baseline, called *Rand_Guess* in the following. Rand_Guess predicts entries following the distribution of the sample set (*Selected* matrix). For example if the ratio of positive:negative entries in a sample set is 1:4, then Rand_Guess would predict an unknown entry as *failed* or *positive* with a probability of 20% and as *ok* or *negative* with a probability of 80%.

3.2.3 Computational cost

CP methods have to be scalable, as they target enormous data sets such as the Netflix database. The computational cost of the optimization problem of learning a MMMF essentially depends on the number of known entries in the *Selected* matrix, or equivalently on the probe fraction. Technically, the optimization is performed through a sparse dual semi-definite program (SDP), with the number of variables equal to the number of observed entries. We used YALMIP [110] as the model tool and CSDP [26] as the SDP solver. Empirically, the time needed for computing one MMMF increases exponentially with the number of entries in the *Selected* matrix. In practice, in our case, computation time was not an issue: less than 30 seconds with 2000 entries (15% probes) on a standard workstation. More generally, the scalability of Matrix Factorization techniques based on low-rank regularizers has been demonstrated many times, and regularizers that converge even faster than the trace norm one have been proposed [100].



(a) lcg-cp (b) srm-ls

Figure 3-5: Accuracy for the Static-Uniform probe selection.

3.3 Experimental results with MMMF

In this section we first show the experiment results of MMMF using three different probe selection strategies, i.e., *static-uniform*, *active probing*, *cost sensitive*. Then AUC orientated optimization is proposed for dealing with the strong imbalance between positive and negative examples. In the end, one implementation of mixed membership matrix factorization which aims at curing the loss of contextual bias is compared with active MMMF on the curated dataset.

3.3.1 Static-uniform

For each result matrix M different fractions of its entries are deleted uniformly and a series of partially observed matrices M'_1, M'_2, ... are generated. For these new matrices, the task is only to predict the deleted entries from the selected ones by MMMF-based CP. Figure 3-5 shows the prediction accuracy as a function of the fraction of launched probes, for the five benchmarks. The results are averages over ten experiments. As lcg-cp and lcg-cr behave similarly, only one is shown. The first and striking result is that an excellent performance can be reached with a tiny fraction of the original probes, typically 5%. The *Rand_Guess* results are plotted for comparison purpose, but can be approximated easily: if q is the fraction of positive entries in the original matrix, then in the deleted part, $P(\text{True Positive}) = P(\text{Positive})P(\text{Predicted Positive}) = q^2$, and similarly $P(\text{True Negative}) = (1-q)^2$; overall, the accuracy is $q^2+(1-q)^2$. With the values of q from Table 3.1, the accuracy of *Rand_Guess*

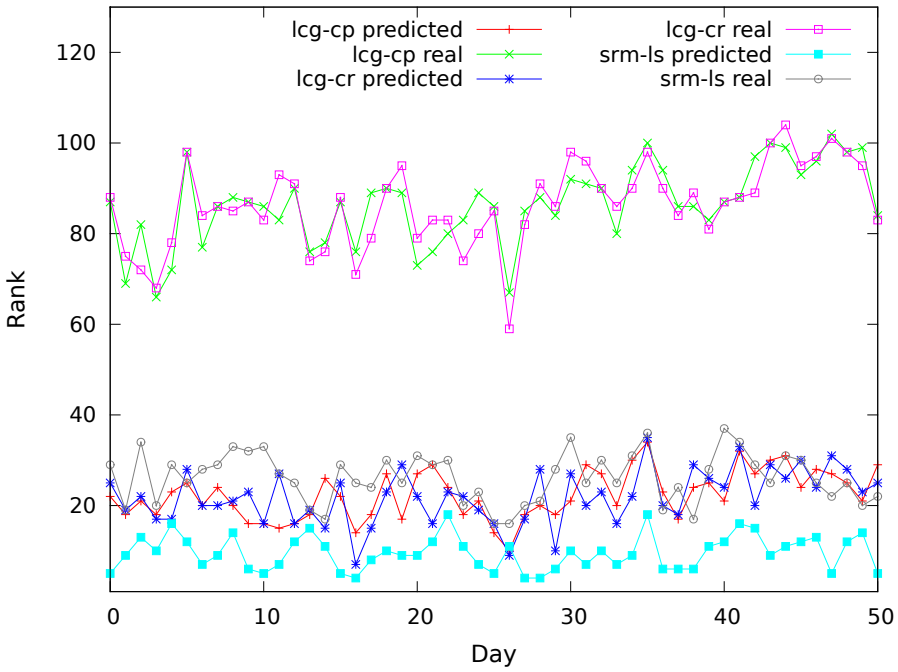
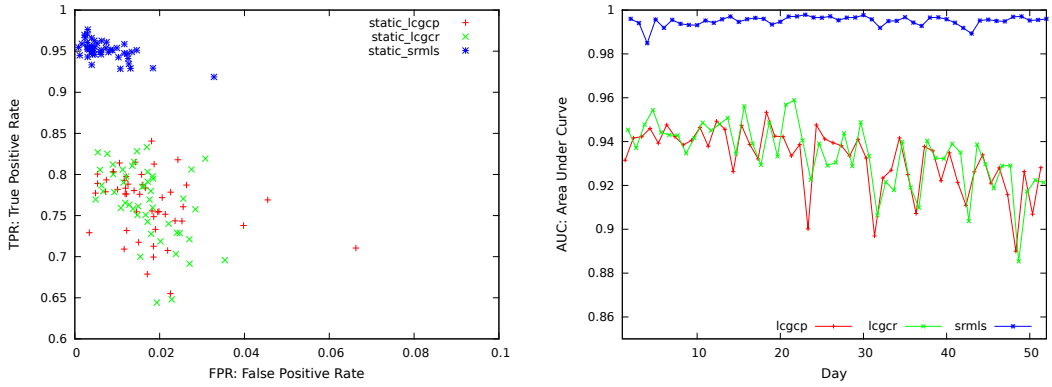


Figure 3-6: Rank comparison for the Static-Uniform probe selection

is in the order of 0.7 for *lcg-cp* and *lcg-cr*, and 0.5 for *srm-ls*.

In the CP interpretation, the rank of a result matrix corresponds to the hidden causes. Figure 3-6 shows the ranks of the predicted and original matrices. The ranks of the predicted matrices are significantly lower than the original ones, showing that a small number of causes dominates the overall behavior. The number of hidden causes is much larger for *lcg-cp* and *lcg-cr* than for *srm-ls*, confirming the empirical evidence that the *srm-ls* faults are more deterministic. However, the rank is far above the theoretical lower bound described in Section 3.1.5. In other words, the sample size is too low to produce a unique solution. By the choice of this low sampling ratio, the method cannot narrow down to the exact factors, and produces a more imprecise and redundant model than the theoretically optimal one.

Figure 3-7(a) is the classical visualization of the confusion matrix in the ROC space for all the 51 days at 90% deletion rate (keeping 10% of the probes). Note the range of the axes, which cover only the small part of the ROC space where the results belong, thus the diagonal line is not visible on the plot. Perfect prediction would yield a point in the upper left corner at coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The *srm-ls* dataset shows excellent



(a) ROC space (partial)

(b) AUC

Figure 3-7: ROC-related metrics, Static-Uniform probe selection

prediction performance, being mostly very close to (0,1); lcg-cp and lcg-cr exhibit close ROC value distributions, definitely much better than a random guess, which lies on the diagonal line. The other indicators also show excellent performance: the AUC (Fig. 3-7(b)) as well as the MCC are close to 1. The case is closed for the initial problem.

The problem becomes much more difficult when the systematic faults are excluded, thus taking the curated matrices as inputs. Figure 3-8 shows the prediction accuracy on the curated *srm-ls* example (figures for the other probes are equally excellent, and are omitted; note that, from Table 3.1, this probe is the most challenging one). As before, at most 10% of the whole probes is needed to reach a promising accuracy, greater than 98%. However, as the number of failed entries left in the curated matrices is much less than in the noncurated ones, e.g., the fraction of failed entries on day 1 (*srm-ls, 04-21-2011*) drops from 45.37% to 2.25%, accuracy is not meaningful: predicting all entries as negative would give a similar result. The ability of making good prediction on the *failed* entries should be valued more. And the relevant performance indicators are not so good, except for day 5, as shown in Fig. 3-10: for the same example (day 1), at 10% deletion rate, sensitivity is 0.32, meaning that 68% of the failures are not predicted, and precision is 0.49, meaning that amongst the predicted failures, 51% are spurious. The first strategy to tackle this issue is Active Probing.

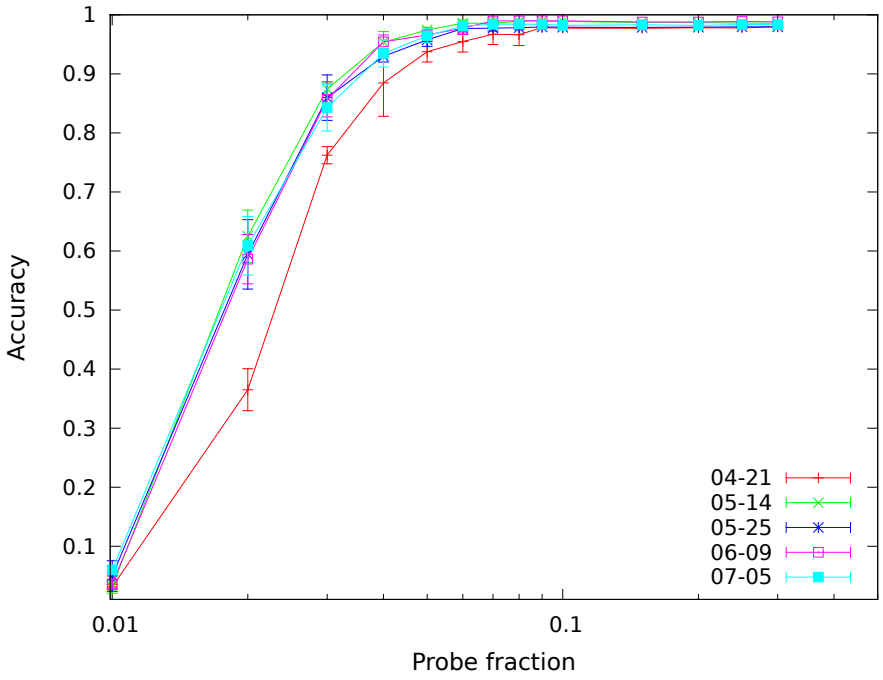


Figure 3-8: Accuracy for the Static-Uniform probe selection, curated *srm-ls*.

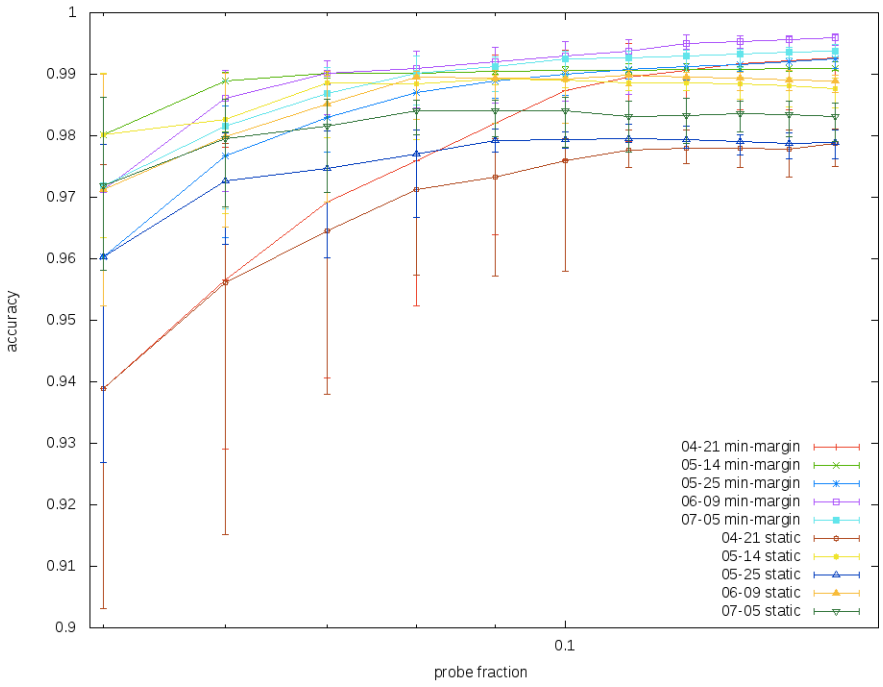


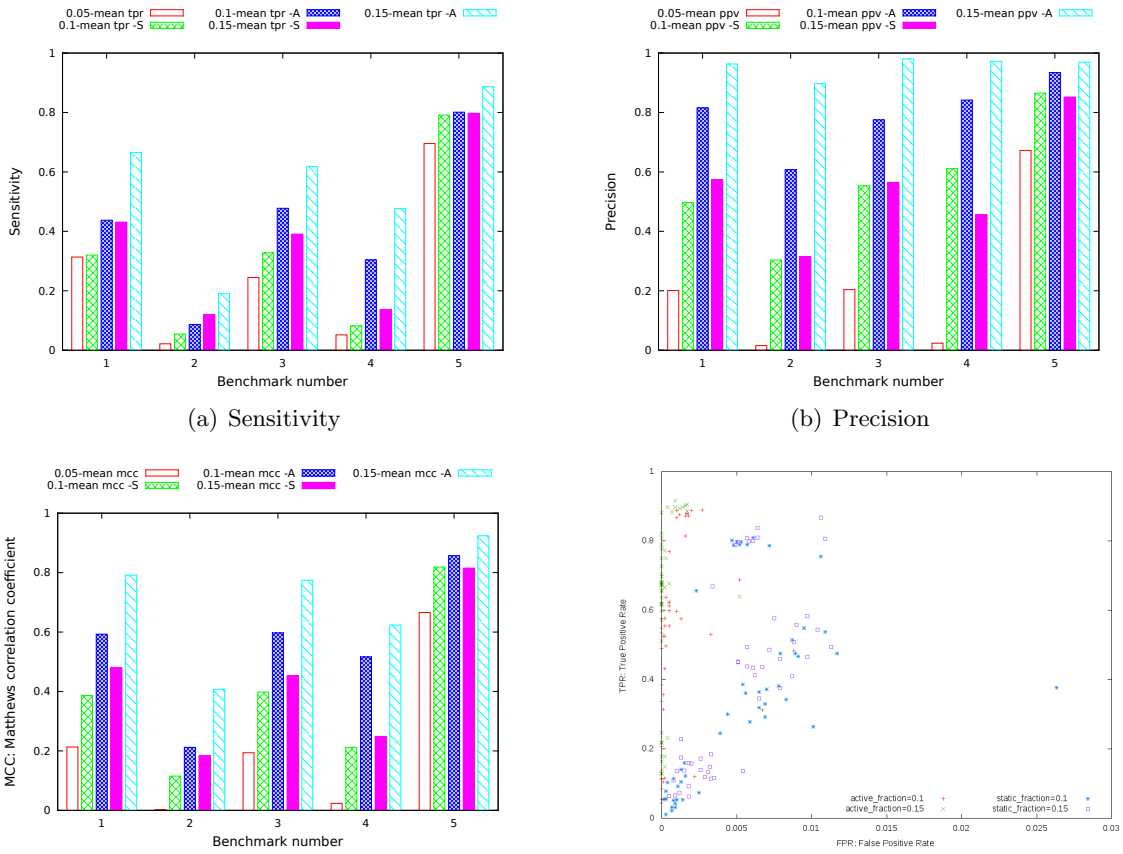
Figure 3-9: Accuracy comparison between the Static-Uniform and Active probe selection, curated *srm-ls* for the five benchmarks

3.3.2 Active probing

In this experiment, we compare the Active Probing strategy with the Static one at equal probing cost: first, a Static-Uniform method is applied, in order to get the reference information, then more probes are selected with the min-margin heuristic for Active Probing, while for the Static-Uniform method, the same number of probes are selected uniformly at random.

Active Probing does improve accuracy over Static-Uniform, as shown in Fig. 3-9. However, as explained in the previous section, the quality of failure prediction is the most important goal in this context. Figure 3-10 compares the relevant indicators: sensitivity, precision and the MCC. They are detailed for the initial probe fraction equal to 5%, then adding probes by step of 5% fractions. The results are given for a total of 10% and 15% probes. The first result is that Active Probing always outperforms Static-Uniform. More importantly, acceptable results can be obtained with a relatively small number of probes (15%), albeit larger than in the much easier noncurated case: in all cases, more than 90% of predicted failures are actual ones (Fig. 3-10(b)), even for the very difficult day 2; the probability of predicting an actual failure (Fig. 3-10(a)) increases from 43% to 67% on day 1, from 39% to 62% on day 3 and from 14% to 48% on day 4. In other words, and as expected, Active Probing singled out the failures as the most uncertain data, adaptively building its own training set.

The performance greatly varies with the benchmark, and the variation is somehow related to the failure rate of the benchmark (Table 3.1): larger failure rates in the original curated matrix help uncovering the structure of the faults, even at quite low levels: with 4% failure rate, the *07-05-2011* (day 5) benchmark exhibits acceptable performance when keeping only 5% of the probes and the Static-Uniform strategy; conversely, for day 2, with a low failure rate (1%), sensitivity remains bad, predicting at best 19% of the actual faults, although active probing allows for a good precision. However, the failure rate does not tell the full story: days 2 and 4 have the same low one, but the performance on day 4 is much better. The likely explanation is that faults on day 2 do not present much correlation, while faults on day 4 derive from a small number of shared causes.



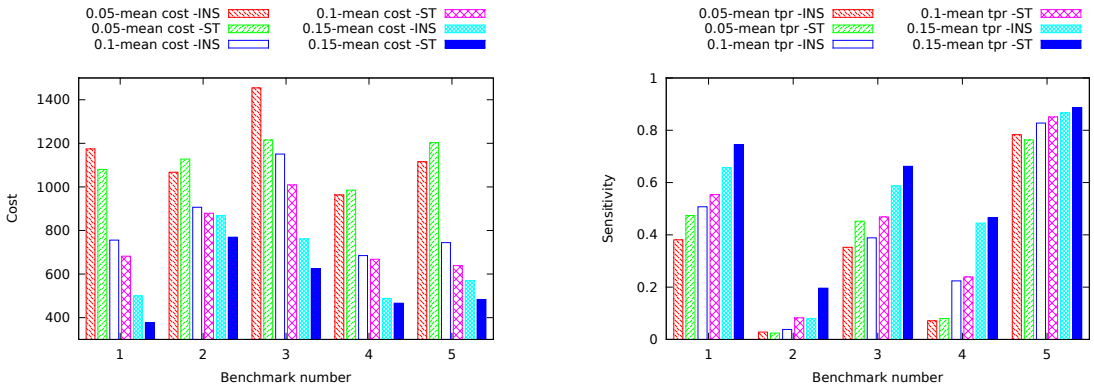
(a) Sensitivity

(b) Precision

(c) MCC

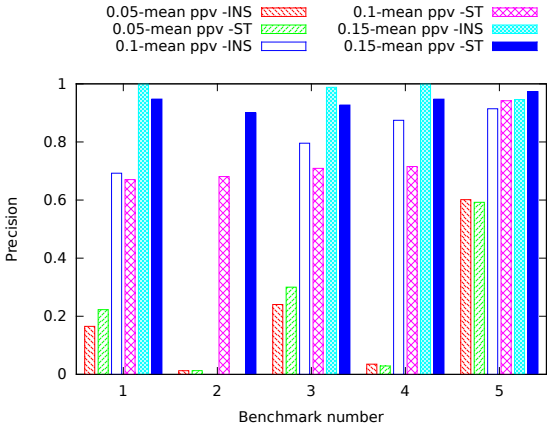
(d) ROC space

Figure 3-10: Performance comparison for Static-uniform and Active Probing, curated *srm-ls*; , (a), (b) and (c) for the five benchmarks, (d) for all days.



(a) Cost

(b) Sensitivity



(c) Precision

Figure 3-11: Cost and performance comparison between the Cost-Sensitive and Cost-Insensitive active probing, curated *srm-ls* for the five benchmarks.

3.3.3 Cost sensitive + Active probing

Finally, we sketch the results of the cost-sensitive MMMF. The C^+/C^- ratio is set equal to 10. The optimization target being soft-margin, the results for the initial Static-Uniform at 5% probe fraction are slightly different from the previous experiments. Figure 3-11 compares Active probing with and without cost weighting. Higher penalization of false negatives almost always decreases the final mis-prediction costs (Figure 3-11 (a)). Figure 3-11 (b) and (c) give the explanation: while sensitivity is indeed increased, the number of false positives also increases, leading to slightly lower precision, but the overall impact is favorable.

3.4 AUC optimization within MMMF

As Section 3.1.5 shows, multiple approaches can be considered within the general framework of Collaborative Prediction. An important experimental result of sections 3.3.1 and 3.3.2 is that static probe selection leaves significant room for improvement, and that adding active learning based on the min-margin heuristic largely fills the performance prediction gap. We need to disentangle as far as possible the contributions of the various ingredients. More precisely, could we remediate to the inefficiencies of the Static Probing MMMF either within the Matrix Factorization framework, or by an altogether different CP approach, while retaining the Static strategy?

In Section 3.3.2, we have exposed the intuitive motivation for Active Probing: static CP applied to fault prediction might underperform because of the strong imbalance between positive and negative examples. This section proposes two static probe selection methods that are optimized towards dealing with this imbalance. If both were to fail, a strong indication would be given towards the role of Active Learning.

MMMF is theoretically grounded only for uniform random selection of examples, which is just the opposite of the active learning approach. Section 3.3.3 shows that the most classical method addressing this issue within the Static Probing strategy provides actual but limited improvement. Thus, a first question is to which extent a more specific algorithm would not successfully compete with Active Probing. We designed one: it integrates MMMF and optimization of the area under the ROC curve (AUC). AUC is a natural and useful performance measure for evaluating classifiers when the class distributions are heavily skewed, because it is invariant to a priori class distribution [28].

Reformulation of the objective function

Intuitively, AUC expresses the probability that a decision function f assigns a higher value to a randomly selected positive example x^+ than to a randomly selected negative example x^-:

$$\mathrm{AUC}(f) = Pr(f(x^+) > f(x^-)).$$

AUC refers to the true distribution of positive and negative instances, and can be estimated through sampling. The normalized Wilcoxon-Mam-Whitney statistic gives the maximum

likelihood estimate of the true AUC given n^+ positive and n^- negative examples [169] :

$$\widehat{AUC}(f) = \frac{\sum_{i=1}^{n^+} \sum_{j=1}^{n^-} 1_{f(x_i^+) > f(x_j^-)}}{n^+ n^-}. \tag{3.6}$$

The AUC score is determined by the number of correctly ranked sample pairs; therefore, to maximize the AUC we could maximize the number of correctly ranked sample pairs, which meet $f(x^+) > f(x^-)$.

Here we extend the standard Maximum Margin Matrix Factorization (MMMF) with the object of AUC optimization. In the MMMF problem, we use the partially observed sparse matrix X to recover Y under the constrain of a L_2 norm discrepancy for each predicted and observed entry, i.e $Y_{ij}X_{ij} \geq 1 - \xi$. However, this entry-wise constraint contains no order information between sample pairs, i.e., order between the pair $\langle X_i, X_j \rangle$ where $i \neq j$ and $X_i \in S^+, X_j \in S^-$. Here, in the aim of AUC score maximization, we add the sample pairwise order constraints to the MMMF objective function and derive the following reformulation.

$$min \;\; \|Y\|_\Sigma + \lambda_1 \sum_{k \in S} \xi_k + \lambda_2 \sum_{i \in S^+, j \in S^-} \delta_{ij} \tag{3.7}$$

$$s.t. \;\; Y_k X_k \geq 1 - \xi_k \tag{3.8}$$

$$Y_i^+ X_i^+ + Y_j^- X_j^- \geq 1 - \delta_{ij}, \tag{3.9}$$

where S is the set of known entries in X, S^+ and S^- are the positive and negative entry sets, ξ_k is the entry-wise constraint on X_k, δ_{ij} is the pairwise order constraint on $\langle Y_i, Y_j \rangle$, λ_1 and λ_2 are the regularization terms. One thing to mention is that the number of constraints in the second regularization term is quadratic with the sample size, thus leading to a more complex optimization problem. Inspired by the idea in [161], instead of adding all constraints at once, we add the most important constraint iteratively, with the price of iterative computation. However, in practice, the total number of added constraints on all test sets proved to be quite limited, never exceeding 12, in accordance with a similar observation in [161].

Algorithm 2: AUC optimization within Matrix Factorization

input : Initial partially observed binary(-1/+1) matrix M_0, max number of iteration N

output : Full binary-valued matrix M^{T_i} predicting unobserved entries of M_0

initialize: Initialize the vars

$S(T_0) = S(M_0)$ /*current constraint set*/ ;

$i = 0$ /*current iteration times*/ ;

while $(i < N)$ **do**

  $M^{T_i} = StandardMC(S(T_i))$ /*Prediction based on observed entries via standard MC procedure*/ ;

  $S'(T_i) = MostViolatedAUC(M^{T_i})$ /*Calculate and select the most violated AUC pair */ ;

  if $\#S'(T_i) > 0$ **then**

    $S(T_{i+1}) = S(T_i) \cup S'(T_i)$;

    $i = i + 1$;

  else

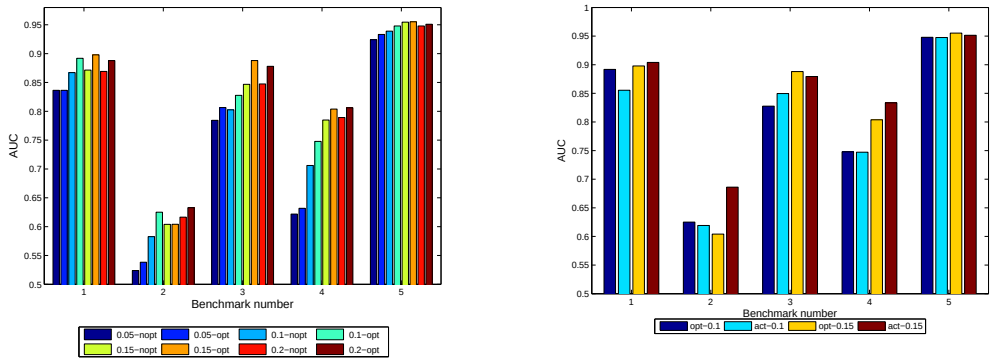
    break /*No violated AUC pair*/ ;

Algorithm Framework

Algorithm 2 illustrates the process of AUC-oriented MMMF (AUC-MMMF). We use a standard MMMF procedure for the recovery of a partially observed matrix, then the AUC value on the training set is computed according to Eqn. 3.6 and the most violated AUC pair is added into the current constraint set for the next iteration. The loop terminates when there is no more violated pairs in the sample set or the maximum number of iteration is reached.

Experimental Results

The trade-off coefficients λ_1 and λ_2 are chosen via cross-validation. We ran each experiment 5 times and average the results correspondingly. The performance of AUC-MMMF with static and active MMMF are compared at different sample rate levels on the curated *srm-ls* benchmarks. Figure 3-12(a) shows that AUC-MMMF always outperforms static MMMF, by about $3\% - 6\%$ on benchmarks $1, 2, 3, 4$. However, Active Probing outperforms or is equivalent to AUC-MMMF in most cases. In other words, the Active Probing strategy actually discovers the violated constraints through focusing on the most uncertain -and very often positive (failure)- cases.



(a) Static vs AUC optimization

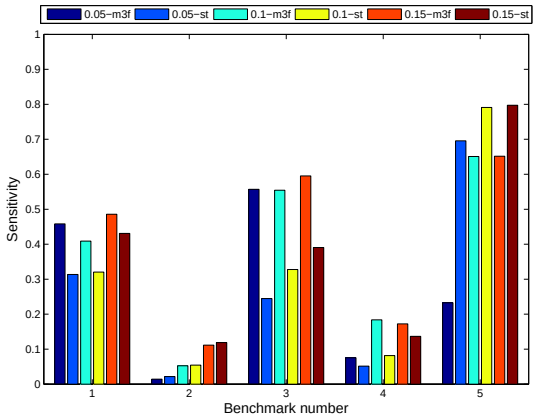
(b) Active vs AUC optimization

Figure 3-12: AUC optimization on *srm-ls*, with $\lambda_1 = 0.5, \lambda_2 = 10$.

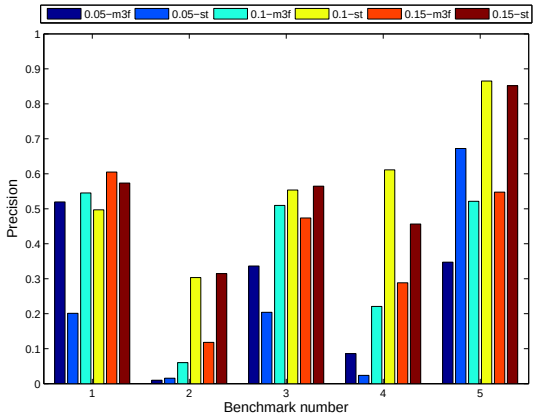
3.5 Mixed membership matrix factorization

As we pointed out in Section 3.1.5, continuous latent factor models such as MMMF produce the predictions in a static way, without considering the contextualization. This might be insufficient when dealing with complex distributed systems, where contextual bias like transient failures and site specific middleware temporary down are frequent. In this section we apply one implementation of the Mixed Membership Matrix Factorization on the curated probing dataset and compare its performance with static and active MMMF. For better understandability, we describe the M^3F Topic-Index Bias (TIB) model of [113] in recommendation terms. The model is an implementation of M^3F where the context bias can be additively decomposed into a user bias and an item bias. Both bias are influenced by counterpart's selected topic, i.e., the user bias is influenced by the item's topic and vice versa. In M^{3}F-TIB each user and each item has its own latent factor vectors (a_u and b_j) and topic distribution parameters (θ_u^U and θ_j^M). To rate an item, first both the involved user and item draw a topic, z_{uj}^U for user side topic and z_{uj}^M for item side topic, from their distributions. Then, a rating bias, β_{uj}^{ik}, is jointly specified by the user and item topics, i and k, and the identity of the user and item, u and j. Last, a complete rating is given by the sum of a user-item-specific static rating $a_u \cdot b_j$ and a contextual bias β_{uj}^{ik}, along with some noise. For simplicity, a rating r can be expressed as following:

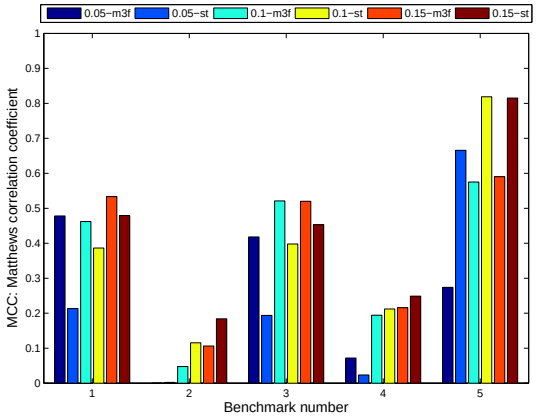
$$r_{uj} \sim N(\beta_{uj}^{ik} + a_u \cdot b_j, \sigma^2),$$



(a) Sensitivity

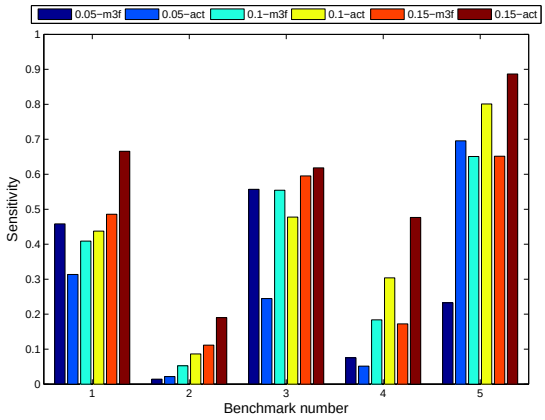


(b) Precision

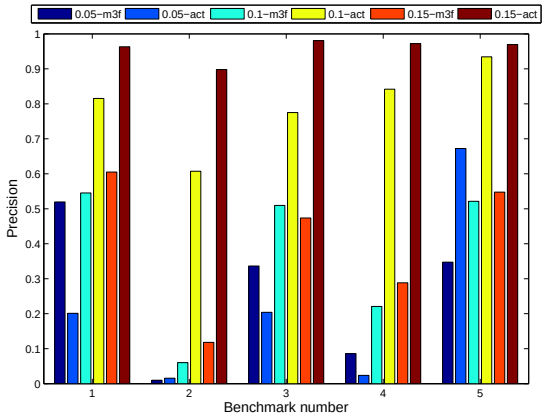


(c) MCC

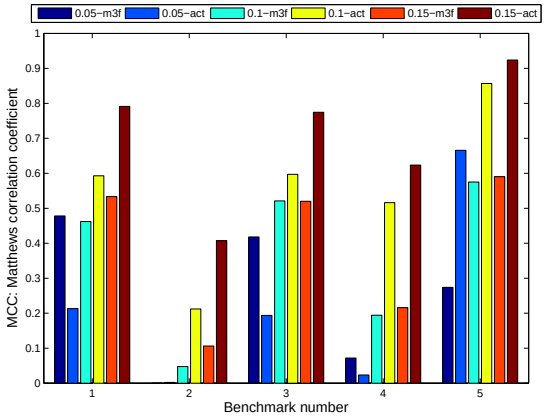
Figure 3-13: M^{3}F-TIB vs. static MMMF, m3f stands for M^{3}F-TIB and st means static MMMF.



(a) Sensitivity



(b) Precision



(c) MCC

Figure 3-14: M^{3}F-TIB vs. active MMMF, m3f stands for M^{3}F-TIB and act means active MMMF.

$$\beta_{uj}^{ik} = \chi_0 + c_u^k + d_j^i,$$

where σ is a Gaussian noise and χ_0 is a fixed global bias, c_u^k is the bias for user u under item topic j and d_j^i is the bias for item j under user topic i.

For M^{3}F-TIB, we used a dynamic threshold for labeling the predicted real-valued matrix, as predicted values provided by M^{3}F-TIB are nearly always negative, which makes a fixed threshold like 0 unreasonable. More precisely, we choose the threshold which assigns the final label of each predicted entry as following: first the proportion of positive samples, $\theta = S^+/S$, is calculated from the training set, then for all the predicted values, we choose the first θ percent as positive entries and the other part as negative ones, in other words assuming that the fraction of positive entries in the training set should approximate the one in the whole set. For M^{3}F-TIB, the model parameters are set as following: $numFacs = 20$, $KU = 2$, $KM = 2$, the Gibbs sampler is initialized using a Maximum A Posteriori (MAP) estimator, and run 500 samples for prediction, without any discarded samples for 'burn-in'. All results are averaged over 10 runs.

Figure 3-13 illustrates the comparison of different classifier measurements between M^{3}F-TIB and static MMMF. On benchmark 1 and 3, M^{3}F-TIB shows a better performance than the static MMMF over all three metrics, except that the precision of M^{3}F-TIB on benchmark 3 is slightly lower than with static MMMF when the size of training set increases. At the same time sensitivity is significantly better than with MMMF, which implies that the dynamic threshold based M^{3}F-TIB tends to have less false negative than false positives. On the other hand, static MMMF performs better than M^{3}F-TIB on benchmark $2, 4, 5$. Interestingly, on benchmark 4, both algorithms tend to have similar MCC values, but behave oppositely on sensitivity and precision: M^{3}F-TIB has better sensitivity while static MMMF has better precision.

Similar comparison of these metrics between M^{3}F-TIB and active MMMF is demonstrated in Fig. 3-14, showing that M^{3}F-TIB is less competitive than active MMMF on all benchmarks. This was to be expected, as M^{3}F-TIB is essentially comparable to static MMMF in our case.

In conclusion, given the operational goal as they are - essentially low intrusiveness probing - the capacity of actively selecting the most informative probes provides a much more efficient method to capture the time variability than M^{3}F.

3.6 Conclusion

The Achille's heel of large scale production grids and clouds is reliability. At the scale of these systems, classical detection or diagnosis would require a complete a priori knowledge of the software and hardware infrastructures that might remain definitely inaccessible. Quality of Experience at reasonable human cost requires extracting the hidden information from monitoring data whose intrusiveness should be limited. Collaborative Prediction is one of the promising and scalable strategies that can address this new challenge. Compared with the recommendation context, monitoring enjoys a decisive advantage, being allowed to adaptively build knowledge. Through experiments on a large dataset, this chapter demonstrates the effectiveness of combining Collaborative Prediction and Active Leaning. The min margin heuristic has shown to be versatile enough to address two difficult issues quite specific to the fault prediction problem, and of very different nature: the spatial one, the imbalance of positive and negative examples; and the temporal one, the transients. Cleaning the data and eliminating the noise, for instance through bias modeling, have been shown essential for recommendation systems. In our active monitoring setting, things go in the reverse way, to the same goal: instead of acquiring all data, then discarding the most noisy of them, which would be for example the result of a fixed frequency probing strategy, Active Probing adaptively adjusts its acquisitions and the internal model. The next step would be to go for a fully personalized recommendation system, taking into account not only the infrastructure, but also the particular user, whose specificities may also create failure risks.

Chapter 4

Sequential Fault Monitoring

This chapter provides detailed discussions on the sequential monitoring of a distributed system. The overall goal is exactly the same as in Chapter 3 : predict the fault behavior with limited probe information. The difference is that we want to take into account the fact that the system dynamically evolves at various time scales. Addressing this issue brings us closer to a model consistent with the practitioners expectations, but turns out to be significantly more difficult than the previous and more idealized setting. Not surprisingly, we cope with this difficulty by exploiting the dynamic setting for enriching the static information presented in the previous chapter with time-related information: *sequential monitoring* deals with a sequence of partially observed matrices and makes prediction using information both from the current and previous time windows. Besides the spatial information preserved in neighboring data points, sequential correlation concealed in consecutive data points is another key feature of sequential data. These sequential patterns are important because of their value which, if exploited properly, can improve the prediction performance of the classifiers.

The main contributions of this chapter are twofold.

- A detailed analysis of the algorithmic alternatives, that contributes to disentangle the components of performance and substantiate the claim that sequential patterns should be exploited;
- An algorithm that efficiently combines the spatial and temporal information sources. Its major strength is to balance exploration and exploitation in a way that formalizes the multi-scale intuition of the practitioner and successfully exploits it.

The organization of this chapter is as follows. Section 4.1 presents the empirical motivation and the formal description of the problem. For clarity, we have structured the rest of the chapter by presenting first all the algorithms that are considered for evaluation, including classical ones and our proposals, followed by the experimental evaluation. Thus, the motivation for proposing new methods will become more evident only in the last two sections. Section 4.2 walks through the various ways to organize spatial and temporal information. Section 4.3 proposes a new algorithm, **SMF** (Sequential Matrix Factorization), and its Active Learning version, **SMFA** (Sequential Matrix Factorization with active learning). Then, Section 4.4 presents a detailed experimental evaluation, conducted on real-world data from the same source as in the previous chapter. The vanilla algorithms are combined in various ways with agnostic optimizations (smoothing) and information-oriented ones (strategies for active learning).

4.1 Introduction

4.1.1 Motivation

In Chapter 3 we have described a method to handle the fault monitoring task of a distributed system based on a collaborative prediction approach. Although this method significantly reduces the number of required probes for acquiring a whole view of the system, it is somehow static. Specifically, the input of this method is a single static matrix describing the current state of the system, without any information from the past.

This setting has two drawbacks. Firstly, it might be appropriate for getting a short time system snapshot; however, due to the ever-changing characteristic of system status it could easily expose its disadvantage when dealing with a long term task. For example, in a large distributed system, there are some stable system components, whose status might be consistent most of the time, while there are also some other components whose status may fluctuate intensely at peak time and remain stable at off-peak time. Second, transient faults are systematically observed: transients are faults that go on and off at high frequency and should be considered as noise; practitioners do not have a clear explanation for them, and they might as well be produced by flaws in the monitoring software itself. Of course, the problem is to disentangle them from real, but short-lived faults.

A further motivation is to explore the possibility of getting rid of Active Learning. In

the previous chapter, we have shown that Active Learning was a required ingredient for efficient prediction in the curated case. On the other hand, Active Learning is somehow inconvenient: it requires a feedback loop and more complicated software than the passive setting. At grid scale, any unnecessary source of complexity should be eliminated. Thus, we explore the hypothesis that the past provides enough information to be equivalent to the one obtained by selectively querying the present.

4.1.2 Categorization

To cope with this sequential fault prediction, a natural extension is to introduce historical information into the framework and to consider fault monitoring as a sequential task. Hence, in this chapter, we reconsider the collaborative monitoring task with a series of time-based inputs and perform the collaborative prediction in a sequential manner.

Generally, there are two types of input information that can be utilized in the sequential case: information from collaborators and information from the past. For simplicity, we call the former one *spatial* information and the latter one *temporal* information. For a component of the distributed system, the spatial information is the current status information of the other components in the system, while the temporal information is its own historical information from the past. In our approach, which is based only on end-to-end probing, the components are the probes' outputs.

Depending on which information a method uses as input, we further divide the methods into three categories.

- The *pure spatial* methods only use information available from the current timestamp; the methods that we described in the previous chapter belong to the pure spatial category. In this chapter, they will be exemplified by **MMMF**.
- The *pure temporal* methods take the entry-wise historical sequences as inputs, independently; they predict entry-wise too, based on time series methods, e.g., Moving Average. In this chapter, they will be exemplified by the **EWMA** method presented in Section 4.2.1.
- The *integrated* methods use both information categories as input. In the classical Collaborative Filtering framework, the so-called *Tensor* method exemplifies this approach and is discussed in Section 4.2.3. Our algorithms, **SMF** and **SMFA** (Section 4.3), fall

into this category. The *collapsed* methods are a subcase - they straightforwardly transfer those employed in static matrix based prediction to the sequential situation and are the simplest approach to integration: first, the temporal information is exploited to build a summary, then a purely spatial method uncovers its hidden structure. In this chapter, they will be exemplified by the **SSVD** method presented in Section 4.2.2.

As our goal is to exploit temporal information, purely spatial methods will be considered here as a baseline; as we have seen before, MMMF has excellent results in the non-curated case and MMMFA exhibits superior performance in the curated case, thus both will be used as a baseline for comparison. The choices that we want to explore are, in increasing order of complexity: purely temporal methods, collapsed methods and fully integrated methods.

4.1.3 Problem statement

We first introduce the formulations and notations of this chapter.

- $X_t \in B^{M\times W}$ denotes the partially observed matrix at time t.
- $\hat{Y}_t \in R^{M\times W}$ is the result of a prediction algorithm.
- $Y_t \in B^{M\times W}$ be the binary version of $\hat{Y}_t$ using a binarization threshold ρ.

The binarized $Y_T(i,j)$ is obtained from $\hat{Y}_T(i,j)$ using a threshold ρ as following:

$$Y_T(i,j) = \begin{cases} -1 \;\; if \quad \hat{Y}_T(i,j) \leq \rho, \\ 1 \;\; \text{otherwise.} \end{cases} \tag{4.1}$$

Since we use '-1' and '1' representing the negative and positive binary values, respectively, if not mentioned, $\rho = 0$ is used as the binarization threshold throughout this thesis.

For a time sequence $1, ..., T$, $X_1, ..., X_T$ are the series of partially observed matrices. We denote this sequence as a cube $Z_T = (X_1, ..., X_T)$, $Z_T \in B^{M\times W\times T}$, thus we have $Z(i,j,t) = X_t(i,j)$. We define the task of sequential matrix prediction in this chapter as: given a partially observed cube (tensor) $Z_t = (X_1, ..., X_t)$, predict the fully estimated matrix Y_t. A simple illustration of this task is shown in Fig. 4-1. As we can see from it, given $X_1, ..., X_t$ as a sequence of partially observed matrices, at each time step t, a matrix Y_t is estimated by the algorithm using information from the observation sequence $X_1, ...X_t$ and estimation sequence of $Y_1, ...Y_{t-1}$. In our proposed algorithms, only Y_{t-1} is used most of the time.

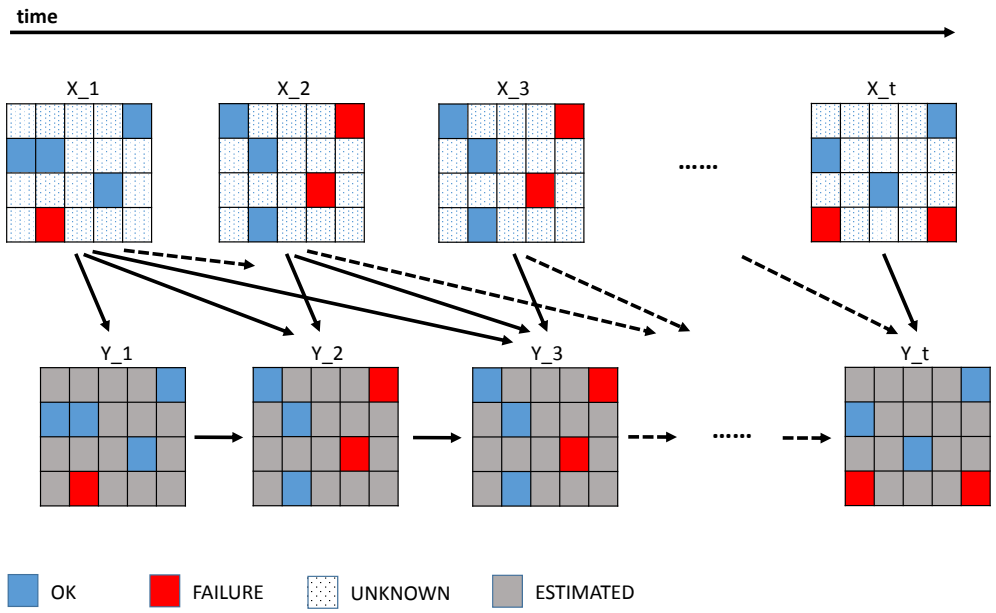


Figure 4-1: An illustrative example of sequential fault prediction problem

With this formalism, temporal methods, that only rely on the entry-wise historical information, try to complete the prediction of $Y_T(i,j)$ only based on information of entry (i,j), which could be only the $X_t(i,j)$, $t = 1, ..., T$ or include $Y_{t-1}(i,j)$, $t = 2, ..., T$. Many commonly used methods like Moving Average(MA), Hidden Markov Model(HMM) can be categorized into this framework.

In spatial methods, where only information of the current time window is available, the prediction of $Y_T(i,j)$ is completed only using knowledge captured in X_T. The underlying assumption of the spatial approach is that for two row-items i_1 and i_2, if they share similar results on a column-item j_1, e.g., $X_T(i_1, j_1)$ and $X_T(i_2, j_1)$ are similar, then they are more likely to have alike results on another column-item j_2, e.g., $X_T(i_1, j_2)$ and $X_T(i_2, j_2)$ are close. Methods like max margin matrix factorization and singular value decomposition are exemplars of this category.

The integrated methods utilize both spatial and temporal information to make a prediction of Y_T. Precisely, to produce a prediction of $Y_T(i,j)$ information encapsulated in X_t and Y_{t-1} for $t = 1, ..., T$ are exploited. The end of integrated methods is to perform a fully exploration and exploitation on both spatial and temporal information. Exemplar methods of this category are tensor factorization based methods and the sequential matrix factorization (SMF) proposed in a later section.

4.2 Background

In this section we first discuss the options for temporal methods, in relation with their intended application to the large scale prediction issue; then present the collapsed approach, followed by a brief survey on the tensor based methods.

4.2.1 Temporal methods

There are extensive types of temporal methods proposed for the sequential prediction, among which auto regressive moving average (ARMA), hidden Markov model (HMM) and exponentially weighted moving average (EWMA) are three commonly used examples. In ARMA, a (weakly) stationary stochastic process is modeled using two polynomials, one for the auto regression and the other for the moving average. Parameters of the AR (auto regression) and MA (moving average) models are learned from the past values of an entry.

In HMM, data series is described by a statistical Markov model, and the system being modeled is assumed to be a Markov process with hidden states. A transition matrix is estimated to give the probability of an entry's value based on its last values in HMM.

In the collaborative monitoring case, ARMA or HMM would need to learn a set of model parameters and update them for each entry in the target (monitored) matrix. This leads to a complexity of $O(KMN)$, where K is the order of a HMM or ARMA model and M and N are the corresponding number of rows and columns in the target matrix, respectively. Computational complexity severely hinders the use of these two methods when dealing with large scale data sets. Thus in this chapter we only consider a simpler temporal method: EWMA, and use it as the base line for comparison with other methods.

Exponential weighted moving average (**EWMA**) [112] is a classical technique in time series, which can be used either for producing smoothed data representation, or for making predictions. Unlike simple moving average where equal weights are assigned to the past observations, exponentially weighted moving average gives exponentially decreasing weights to data over time.

$$y_t = \begin{cases} x_1, & t = 1, \\ \theta x_t + (1-\theta) y_{t-1}, & t > 1 \end{cases}, \tag{4.2}$$

where $\theta \in (0, 1)$ is a damping factor used to control the impact of past data, x_t is the observation at time t and y_t is the EWMA value at time t. A higher θ would discount the older observations faster.

In the sequential monitoring task, at a time step T the way we use EWMA to produce an estimation for an entry (i, j) of its current status based on its history observations is as following:

$$\hat{Y}_T(i,j) = \begin{cases} X_1(i,j), & T = 1, \\ \theta X_T(i,j) + (1-\theta)\hat{Y}_{T-1}(i,j), & T > 1 \end{cases}. \tag{4.3}$$

Note that in Eqn. 4.3, $\hat{Y}_T(i, j)$ is estimated based on information only from the entry's past observation sequence $X_t(i, j), t = 1, ...T$, making it a pure temporal method. For simplicity, in the following we keep the EWMA name for the point wise application of EWMA to the matrix entries.

Steps of using EWMA to estimate Y_T are given in Algorithm 3. The last L time window sample sequence $X_{T-L+1}, ..., X_T$ is used as input for EWMA. However, it is possible that all of these are missing, that no probe has been launched on this particular (i, j) pair in the window: for instance, if h_2 is uniform over space with a 10% selection rate and $L = 20$, the probability of a missing entry is $\sim 12\%$. In this case, the missing entry in $\hat{Y}_T$ is filled with the corresponding entry in $\hat{Y}_{T-1}$, as the best available estimate. For ensuring a starting point, at the initial stage, Y_{init} is computed as the exponentially weighted moving average of a sequence of fully observed matrices with length L, i.e., $X_1, X_2, ..., X_L$. Because $X_1, X_2, ..., X_L$ are fully observed matrices, no missing entry exists in Y_{init}.

Algorithm 3: Exponentially Weighted Moving Average, EWMA

Input: N, number of random samples;
Y_{init}, initial value for each entry;
ρ, threshold for binarization;
$X_{l,l=t-L,...,t-1}$, history sample sequence;
Output: Full binary-valued matrix Y_t
Initialize: $X_t \leftarrow 0$, $Y_1 \leftarrow X_1$, init random sample heuristic h_2

$S_r \leftarrow Sample(h_2, N)$ /*Select N random sample indexes*/;
$X_t(S_r) \leftarrow QueryLabels(S_r)$ /*Query the true labels for entries in S_r*/;
$\hat{Y}_t \leftarrow \theta X_t + (1 - \theta)\hat{Y}_{t-1}$;
$I \leftarrow$ findMissingEntries($\hat{Y}_t$) ;
for $i \in I$ **do**
    $\hat{Y}_t(i) \leftarrow \hat{Y}_{t-1}(i)$
$Y_t \leftarrow binarization(\hat{Y}_t, \rho)$ /*Turn the real-valued Y_t into Binary matrix*/;
return Y_t

4.2.2 Collapsed methods

Principle

The idea behind collapsed methods is to exploit a full matrix built by a temporal method: predictions are produced based on a full matrix X'_t. Going back to the essential intuition that we are looking for low-rank approximation, dimension reduction methods are then employed on X'_t for matrix factorization. Thus, in this section, we test the adaptation of Singular Value Decomposition to the the sequential case under the name of sequential singular value decomposition, **SSVD** (mentioned as Truncated SVD in [3]).

Sequential Singular Value Decomposition, SSVD

The rank-R SVD approximation of a matrix X is given by:

$$X \approx U_R \Sigma_R V_R^T \approx \sum_{k=1}^{R} \sigma_k u_k v_k^T, \tag{4.4}$$

where V^T is the transpose of V. Given a partially observed matrix X_t, the way we predict Y_t by using **SSVD** is described in Algorithm 4. In the first step of SSVD, missing values in X_t are imputed using the method *ImputeMatrix* (described in Algorithm 5). An EWMA imputation is implemented in *ImputeMatrix* which replaces the missing entries using an exponentially weighted moving average of its past values. Besides this EMWA based imputation, other alternatives like KNN imputation, column (row) wise imputation for missing entries in a matrix are also commonly used methods. In the second step of SSVD, a SVD decomposition is employed on the imputed matrix and then the top R-rank approximation is binarized and returned as the estimated matrix of SSVD.

Discussion

Another possible collapsed method would be to build a partial matrix X'_t from the past, but including only the actually observed past entries, and then perform matrix completion, e.g., MMMF, on the not so sparse X'_t. As we mentioned in Chapter 3, the MMMF method regards each observed entry in X as a constrained variable in the prediction process, and the computational complexity increases drastically with the number of involved constraints. Therefore, it is not appropriate to use a MMMF method on the collapsed matrix.

4.2.3 Tensor factorization

A number of approaches have been proposed for the sequential matrix completion problem based on either tensor or matrix factorization. For the recommendation application, [101] splits the sequential data into several coarse time domains. It then assumes a static group-level rating distribution and a slightly drifting individual user interests across the time domains. A cross-domain CP framework is used to share the static group-level rating matrix, together with a Bayesian latent factor model for capturing the drifting behavior of an individual user. The inference model is learned using Gibbs sampling. This method

Algorithm 4: SSVD, sequential R-rank SVD approximation

Input: N, number of random samples;
R, # of top singular components to select;
ρ, threshold for binarization;
l, # of past observations used for imputing missing entries;
Output: Full binary-valued matrix Y'_t
Initialize: $X_t \leftarrow 0$

$S_r \leftarrow Sample(h_2, N)$ /*Select N random sample indexes*/;
$X_t(S_r) \leftarrow QueryLabels(S_r)$ /*Query the true labels for entries in S_r*/;
$X'_t \leftarrow ImputeMatrix(X_t, X_{t-l,...,t-1})$ /*Impute missing entries in X_t;
$[U, \Sigma, V^t] = SVD(X'_t)$ /*SVD decomposition of X'_T*/ ;
$\hat{Y}_t \leftarrow U_R \Sigma_R V_R^t$ /*Top R-rank approximation*/ ;
$Y'_t \leftarrow binarization(\hat{Y}_t, \rho)$ /*Turn the real-valued Y_t into Binary matrix*/;
return Y'_t

Algorithm 5: ImputeMatrix, impute missing entries in a matrix

Input: X_t, matrix with missing entries ;
$X_{t-l,...,t-1}$, matrix sequence for imputing missing entries;
Output: X', imputed matrix

$X' \leftarrow X_t$;
$I \leftarrow$ findMissingEntries(X_t) /*find missing entries in X_t*/ ;
for $i \in I$ **do**
    $X'_{t-l} \leftarrow X_{t-l}$;
    for $j = t - l + 1, ..., t - 1$ **do**
        $X'_j(i) \leftarrow \theta X_j(i) + (1 - \theta) X'_{j-1}(i)$ /*Impute missing entries via EWMA*/
$X(I)' \leftarrow X'_{t-1}(I)$;
return X'

is suitable for modeling relative long term (coarse time domains) user interests, however, not appropriate for capturing system transients. A user's interest will certainly last for a relatively long time, but a component's status in a complex system may fluctuate frequently. [106] extends the low-rank matrix completion to the tensor case by proposing the trace norm for tensors. As in the matrix completion case, the tensor completion is formulated as a convex optimization problem, and is solved by three heuristic methods proposed by the user. A recent method concerning sequential active matrix and tensor completion is proposed in [96]. The proposed algorithms in this paper employ adaptive sampling schemes to obtain strong performance for the low-rank matrix and tensor completion problem. Entries which are informative for learning the column space of the matrix (tensor) are identified through an adaptivity exploitation. Theoretical results of the sufficient number of adaptively selected

samples for an exact recovery are given both for the matrix and tensor case. See [63] and [92] for additional research on this topic. Despite extensive alternatives, in this thesis, we only test the raw tensor factorization method since it's simple, easy to implement and can serve as a base line for the tensor based approaches. In the following, we will firstly go through the basics of tensor factorization, and then discuss its applicability for our problem.

A tensor is a multidimensional array. A N-way (or Nth-order) tensor can be described as the product of N vector spaces. This decomposition can be used to reveal underlying linear structures in the data, and has applications like noise reduction or data compression. Generally speaking, two particular tensor decompositions are widely discussed: CANDECOMP/PARAFAC (CAPA for short) and Tucker [92]. The CAPA decomposes a tensor as a sum of rank-one tensors, and the Tucker decomposes a tensor into a set of matrices and one small core tensor. For higher dimensional N-way tensor (like the 3-way monitoring dataset), we could use the tensor decomposition technique straightforwardly, without any preprocessing like collapsing the tensor into a flat matrix.

The benefits brought by not collapsing the tensor data into a flat matrix but keeping its natural high dimensional structure are two fold: firstly, the underlying patterns in multi-way datasets are preserved. Because the collapsing of data along any dimension will cause a loss of information in that dimension, it is therefore beneficial to keep the natural structure as much as possible. Secondly, CAPA yields a highly interpretable factorization that includes a time dimension, and patterns in the time dimension can be extracted out straightforwardly. Unlike matrix based prediction which is limited to predict for a single time step, CAPA can be used in both single step and periodic temporal prediction problems.

Compared with the Tucker decomposition, the CAPA model is more advantageous in terms of interpretability, uniqueness of solution and determination of parameters [29]. A CAPA mode-3 decomposition can be expressed as either a sum of rank-one tensors (each of which has an outer product of vectors a_r, b_r, c_r and a weight λ_r) or factor matrices:

$$Z \approx \sum_{r=1}^{R} \lambda_r (a_r \circ b_r \circ c_r) \equiv [\lambda; A, B, C], \tag{4.5}$$

where Z represents the raw data tensor and R specifies the number of rank-one components. $a \circ b$ means the outer product of a and b. Take the sequential monitoring for example, we have a partially observed third-order tensor Z_T, $Z \in B^{M,N,T}$ at time T, and the CAPA tensor

factorization decomposes Z_T into a set of rank-one components. Then the missing entries in Z_T can be recovered by making an outer product of the first R rank-one components.

Theoretically, the tensor factorization based method is able to deal with sequential matrix factorization in a promising way. Because information along the temporal dimension is processed straightforwardly without any collapse, the temporal transition can be preserved in the factorization. However, practically, tensor factorization without any regularization can be seen as a simple linear regression exerted along each dimension, thus only those principally important factors are kept in the result. It is more appropriate to introduce regularization on some specific observations (e.g., positive in our case), and orientate the regression as we want. Moreover, entries are equally weighted in a tensor factorization method, which might impede the method's suitability, especially in the sequential fault prediction case where the most recent information should weight more than the far past one.

4.3 Sequential matrix factorization

4.3.1 SMF algorithm

As mentioned before, there are two types of information available in the sequential monitoring: spatial and temporal information. The spatial information can be thoroughly exploited by a collaborative prediction method like MMMF, while on the other hand, the temporal information which concerns the entries' evolving characteristics provides extra opportunity for improving the prediction performance. Specifically, at each time step t we have a sequence of history predictions $Y_1, ..., Y_{t-1}$ and for each estimated value in $Y_{i,i=1,...,t-1}$ we could measure our confidence in prediction as each entry's distance to the separation hyperplane. Thus two types of predictions emerge: those predictions close to the separation plane and those far from the separation plane. We call the former one the *most uncertain* predictions and the latter the *most confident* predictions. From the system point of view, the most uncertain predictions are related to those components with short term status like the transient faults. On the other hand, the most confident predictions are related to those components with relatively long term stable status. Hardware permanent failures or middleware deployment bugs may create the most stable faults; at the intermediate time scale, some software components that may be significantly malfunctioning when overloaded would

provide a relatively stable result.

In this section, we propose an algorithm, sequential matrix factorization (SMF), to utilize both the spatial and temporal information, such that both the long term and short term status behavior can be exploited. In the following discussion, we use S_u, S_c and S_r as the index set in matrix X, where each of them denotes the most uncertain prediction set, most confident prediction set and the random sample set, respectively. Recall that the objective function of MMMF is:

$$\arg\min_{Y} ||Y_t||_{\Sigma} + C\mathcal{L}_h(Y_t(S_r), X_t(S_r)), \tag{4.6}$$

where C is a regularization coefficient, S_r is the randomly sampled set in X_t, and $\mathcal{L}_h(Y_t(S_r), X_t(S_r))$ is the hinge loss between Y and X defined as:

$$\mathcal{L}_h(Y_t(S_r), X_t(S_r)) = \sum_{ia \in S_r} max(0, 1 - Y_{ia}X_{ia}). \tag{4.7}$$

The objective function (Enq. 4.6) is composed of two terms, where the first one is the trace norm of the estimated matrix Y_t and the second term is the discrepancy between estimation and observation. In the following we will develop the objective function of SMF by adding the *most uncertain* and the *most confident* information to Eqn. 4.6, incrementally.

First we consider the most uncertain information. Similar to the *most uncertain* heuristic applied in the active probing in Section 3.3.2, in the sequential case the most uncertain prediction set S_u (entries with small margin to the classification hyper-plane) can be derived from Y_{t-1} and their labels at time t can be queried from the system. Hence, the ground truth of those most uncertain predictions in Y_{t-1} are available in the sample set X_t. We denote this as $X_t(S_u)$.

The second information is the *most confident* predictions concealed in the history estimation. For these most confident entries S_c, instead of sampling their true labels at time t, their previous predictions can be used straightforwardly in the next run. Specifically, in SMF we choose those most confident predictions from Y_{t-1} and assume their states remain unchanged at time t with a consistency level γ. We compute γ in terms of the difference between Y_{t-1} and X_t, i.e., difference between last estimation and current observation. Typical classification criteria like sensitivity (TPR) or FSCORE can be used for measuring this

consistency. The most uncertain prediction set S_u plus the random sample set S_r constitute the observed set in X, i.e., $X(S_u \cup S_r)$. The consistency level γ can be calculated between $X_t(S_u \cup S_r)$ and $Y_{t-1}(S_u \cup S_r)$ as following:

$$\gamma = TPR(Y_{t-1}(S_u \cup S_r), X_t(S_u \cup S_r)), \tag{4.8}$$

where $TPR(A, B)$ is a function for computing the sensitivity (true positive rate) of A according to the ground truth set B. In the prediction, γ is used as an adaptive cost ratio which adjusts the weight (penalty) of the heuristic information in the objective function (similar to the coefficient C in Eqn. 4.6). The reason we choose TPR as the penalty lies in the fact that in distributed system monitoring a successful discovering of a failure becomes more important than an false alarm.

In addition to the most uncertain set S_u and most confident set S_c, we also introduce a random set S_r in the objective function of SMF. It serves as a term for avoiding over-fitting in the history information, where sudden change between the last estimation and the current observation might occur. To sum up, the SMF has the following formula:

$$\arg\min_{Y} ||Y_t||_\Sigma + C\mathcal{L}_h(Y_t(S) - X_t(S)) + C\gamma\mathcal{L}_h(Y_t(S_c) - Y_{t-1}(S_c)), \tag{4.9}$$

where $S = S_u \cup S_r$ is the sample set we query labels at time t and S_c is the most confident prediction set we inherit from $t-1$. The difference between Eqn. 4.6 and 4.9 is exhibited by the selection of S_u and the presence of S_c. As is the same in Eqn. 4.6, Eqn. 4.9 is also convex, and the method we have described in Chapter 3 can be used to find the global minimum directly.

Figure 4-2 illustrates the basic idea of the heuristics mentioned above. The most uncertain and most confident predictions are selected from Y_{t-1}, where labels of the former set are further queried at time t, and labels of the latter set are inherited from the corresponding estimation values in the last run.

Algorithm 6 describes the pseudocode of SMF. At the beginning, the sample set S of X_t is generated by a combination of selecting the most uncertain predictions from Y_{t-1} and a random sampling (line 1 to 3). Then the true labels of S are quired from the system and are used as ground truth for evaluating the discrepancy between Y_{t-1} and X_t (line 4, 5). The most confident predictions in Y_{t-1} are selected in the following step and used as input

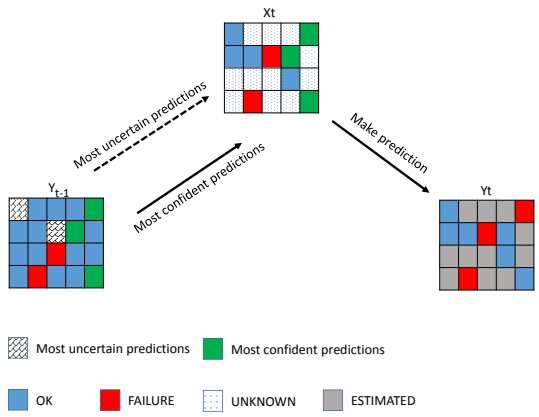


Figure 4-2: Illustration of heuristics in SMF

for the current estimation. In the final step Y_t is derived by finding an estimation which minimizes Eqn. 4.9.

4.3.2 Sequential matrix factorization with active sampling

Like in the active matrix factorization, another intuitive way to improve the sequential prediction performance is to choose the sample entries in X_t actively and iteratively. In SMF, there are three ways to select sample entries: random, most uncertain and most confident. The latter two strategies rely on information from the last prediction Y_{t-1}. The selection of active samples is completed all at once in SMF. No further actions are taken after its first estimation of Y_t^1 [1]. However, heuristic information like most uncertain and most confident prediction also exist in Y_t^1 or its following estimations $Y_t^i, i = 2, 3, ...$ If exploited properly, it can also profit the estimation. For example, samples in the active MMMF are selected using the most uncertain heuristic from the last prediction iteratively and actively until the maximum number of samples is reached. With the progress of each iteration, confidence as well as prediction performance in the estimation increase simultaneously.

Based on the above observation, in this section we propose the sequential matrix factorization with active learning (SMFA) based on an iterative use of SMF. Steps of this algorithm are described in Algorithm 7. For simplicity, we denote the estimation matrix

[1] Here we use Y_t^i to denote the estimation at time t in the ith iteration

Algorithm 6: Sequential Matrix Factorization (SMF)

Input: $\hat{\mathbf{Y}}_{\mathbf{t-1}}$, last prediction;
$\mathbf{N_u}$, number of most uncertain samples from Y_{t-1};
$\mathbf{N_c}$, number of most confident samples from Y_{t-1};
$\mathbf{N_r}$, number of random samples;
$\mathbf{C}$, slack penalty.

Output: Full real-valued matrix $\hat{\mathbf{Y}}_{\mathbf{t}}$

Initialize: $Init\ h_1, h_2, h_3$, /*Initialize the most uncertain, most confident and random sampling heuristic, respectively*/;

1 $S_u \leftarrow Sample(h_1, N_u, \hat{Y}_{t-1})$ /*select N_u most uncertain sample indexes from $\hat{Y}_{t-1}$*/;
2 $S_r \leftarrow Sample(h_2, N_r)$, /*select N_r random sample indexes*/;
3 $S \leftarrow S_u \cup S_r$;
4 $X_t(S) \leftarrow QueryLabels(S)$, /*query the true label for entries in S*/ ;
5 $\gamma \leftarrow TPR(X_t(S), Y_{t-1}(S))$ /*given $X_t(S)$ (true labels for entries in S), compute the sensitivity of $Y_{t-1}(S)$*/;
6 $S_c \leftarrow Sample(h_3, N_c, Y_{t-1})$, /*select N_c most confident samples from Y_{t-1}*/;
7 $\hat{Y}_t \leftarrow \arg\min_Y ||\hat{Y}_t||_\Sigma + C\mathcal{L}_h(\hat{Y}_t(S) - X_t(S)) + C\gamma\mathcal{L}_h(\hat{Y}_t(S_c) - \hat{Y}_{t-1}(S_c))$ /*find an estimation that minimizes the objective function*/;
8 **return** $\hat{Y}_t$

at time t of the ith iteration as Y_t^i. At the beginning, we use the SMF to get an initial estimation Y_t^0 from Y_{t-1} (line 4), then an iterative estimation is employed on the prediction sequence $Y_t^i, i = 1, 2, ...$ until the maximum number of samples is reached (line 5 to 9). Active sample selection is engaged each time the SMF algorithm selects the most uncertain and most confident predictions from the last estimation.

4.3.3 Smoothing the outputs

Although one of the key features in SMF or SMFA is to preserve the continuity of predictions between consecutive time windows, this can still be enhanced by smoothing the output sequence of a method. Smoothing method, e.g., EWMA, can be applied on the output sequence: $Y_1, Y_2, ..., Y_t$, such that the prediction is relatively consistent in time. Smoothing the prediction sequence using EWMA can be achieved as follows:

$$Y'_k(i,j) = \begin{cases} Y_k(i,j), k = t - l + 1, \\ \theta Y_k(i,j) + (1-\theta)Y'_{k-1}(i,j), k = t - l + 2, ..., t \end{cases}, \tag{4.10}$$

where $\theta \in (0, 1)$ is an user defined damping factor, and l is the lag window length.

Algorithm 7: Sequential Matrix Factorization with Active sampling (SMFA)

Input: N, max # of new samples;
$\hat{Y}_{t-1}$, last prediction;
P_0, initial sample rate for the 1st prediction;
P_a, active sample rate at each iteration;
ρ, ratio of random samples and most uncertain samples for P_a;
C, slack penalty.

Output: Full real-valued matrix $\hat{Y}_t$

initialize: $Init(N_c)$ /*Initialize the number of most confident samples to select in each iteration*/;

$i = 0$ /*current iteration index*/ ;

$n = N \times P_0$ /*current number of new samples*/ ;

$[N_u, N_r] \leftarrow getSampleSize(n, \rho)$ /*Get random and most uncertain sample size for the initial prediction*/;

$\hat{Y}_t^i \leftarrow SMF(\hat{Y}_{t-1}, N_u, N_c, N_r, C)$;

while $(n < N)$ **do**

  $[N_u, N_r] \leftarrow getSampleSize(N \times P_a, \rho)$ /*Get random and most uncertain sample size according to ρ and P_a*/;

  $\hat{Y}_t^{i+1} \leftarrow SMF(\hat{Y}_t^i, N_u, N_c, N_r, C)$;

  $n = n + N_u + N_c + N_r$;

  $i = i + 1$;

$\hat{Y}_t = \hat{Y}_t^i$;

return $\hat{Y}_t$

4.3.4 Summary

In the previous sections, we have introduced several methods concerning the application of sequential fault monitoring. We have given a brief summary on their input, output data and related parameters in Table 4.1. In addition to the raw methods, we also proposed a smoothing version for each of them. For a method A, we mark its smoothed version as A* (e.g., the smoothed version of SMF is marked as SMF* in later section).

Table 4.1: Summary of sequential methods

	Input Data	Output Data	Parameters
EWMA	$X_{t-L+1}, ..., X_{t-1}, X_t$	Y_t	N, # of samples; θ, damping factor; L, lag window length.
SSVD	X_t	Y_t	N, # of samples; R, rank of SVD approximation; L, lag window length for imputation.
MMMF	X_t	Y_t	N, # of samples; C, coefficient for slack penalty; Σ, norm used for distance measurement.
SMF	X_t, Y_{t-1}	Y_t	N_r, # of random samples; N_c, # of most confident samples. N_u, # of most uncertain samples; C, slack penalty.
SMFA	X_t, Y_{t-1}	Y_t	N, # of total samples; C, slack penalty; P_0, initial sample rate; P_a, active sample rate at each iteration; ρ, ratio of random sample and most uncertain sample for P_a;
TENSOR	$X_{t-L+1}, ..., X_{t-1}, X_t$	Y_t	N, # of samples in X_t; R, # of rank-1 components; λ, $R \times 1$ vector, with each one be the weight of an outer product of a sub-dimension; L, lag window length for factorization.
MMMFA	X_t	Y_t	N, # of total samples; C, slack penalty; P_0, initial sample rate; P_a, active sample rate at each iteration; ρ, ratio of random sample and most uncertain sample for P_a;
A*	$Y_{t-L+1}, ..., Y_t$	Y'_t	L, lag window length; θ, damping factor for smoothing.

4.4 Experiments

This section provides a view of the result of previously proposed algorithms on a real-world dataset. Similar to the dataset used in Chapter 3, this one is also collected on the EGI grid infrastructure, describing the status of distributed gird CEs and SEs in a time series.

4.4.1 Data description

The dataset was collected on the EGI by submitting a series of jobs to 212 CEs every two hours between *Mon Nov 12 15:52 CET 2012* and *Sat Nov 24 09:54 CET 2012*, which is about 282 hours in total. The goal of these jobs was to collect service availability information between CEs and SEs. Specifically, in this experiment the probe *lcg-cp* was launched from each CE to test its connection to all SEs. 96 SEs were tested every two hours from each CE. To make the validation of the algorithm more convincing, we remove those CEs with less than 7000 observed entries and those time windows with less than 50% observations from the dataset. The final data is a cube of size $79 \times 96 \times 119$, with each dimension corresponding to CE, SE, and time window, respectively.

The goal of our experiment is to predict whether the jth SE is accessible from the ith CE at a given time window t. We use 0 to represent a missing observation, 1 for a *Failure* connection, and -1 for an *OK* connection. This notation is in accordance with the general meaning of positive (abnormal) and negative (normal) in statistics.

For simplicity, let M be the total number of CEs, W be the total number of SEs, and $t_{k,k=1,2,\ldots,T}$ be the time window sequence, we further note N_{t_k} as the number of observed entries at t_k and $N^+_{t_k}$ be the number of positive entries (failures) at t_k, then the observation rate and test failure rate at t_k is defined as:

$$r_{t_k} = \frac{N_{t_k}}{M \times W},$$

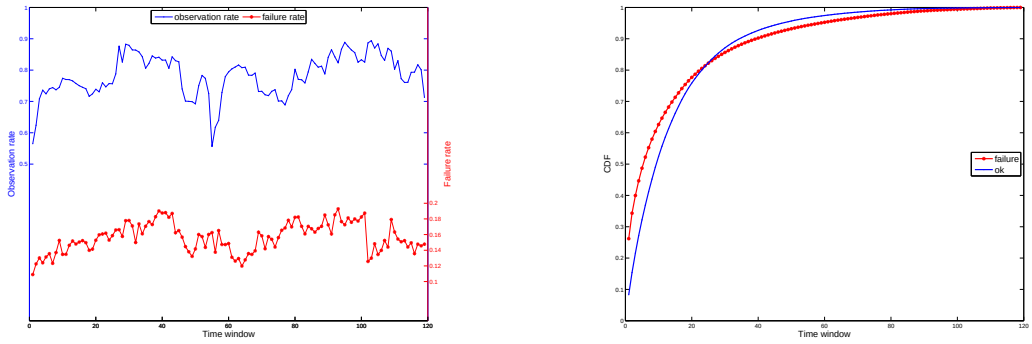
and

$$f_{t_k} = \frac{N^+_{t_k}}{M \times W},$$

respectively. Figure 4-3(a) illustrates the observation rate (r_{t_k}) and failure rate (f_{t_k}) of the dataset, where most of the observation rates stay above 70% and the failure rates are less than 20%. A high observation rate ensures a more reliable result when comparing different algorithms since we have more ground truth information at hand, on the other hand, a relatively stable failure rate implies a consistent system status in consecutive time windows. However, as shown in the figure there are several sudden changes of the failure rate presented in the data, e.g., the sharp drop from 18.74% to 12.57% at the 101st time window. Their impact on the algorithm performance is discussed in later experiments.

sequence	1	1	0	1	1	1	-1	0	-1	-1
	↑	↑	↑	↑	↑	↑	↑	↑	↑	↑
duration	2	1		3	2	1	1		2	1

Table 4.2: Illustration of duration length for OK and Failure



(a) Observation rate and failure rate over time

(b) CDF of duration of OK and Failure

Figure 4-3: Statistics of dataset.

Another interesting aspect of the data is the duration distribution of each system status (i.e., OK or Failure). The duration length of a system status is defined as the number of time windows the status spans until a different status is observed in the sequence. Suppose we have a sequence $a = \{1, 1, 0, 1, 1, 1, -1, 0, -1, -1\}$. An illustrative example of the duration length caculation for OK and Failure is displayed in Table 4.2. After computing the duration length of each observed entry, we compute the CDF of it and show the result in Fig. 4-3(b). As we can see, the proportion of duration length 1 for failure is about 25%, while this number for OK is about 9%. Both failure and OK have about 80% entries with a duration length around 30. Since missing entries are neglected during the caculation this measurement is actually a lower bound estimation of the duration length of OK and Failure. In other words, the real duration length should be longer if we take into account the missing entries.

To conduct the comparison, two versions of the dataset are used in the experiments: the raw dataset and the curated dataset. As before the curated one is the same as the raw dataset except those rows or columns with more than 98% failed entries are removed

(column or row with 100% failed entries does not exist in this dataset). In the sequential setting, testing algorithms on the raw dataset is as meaningful as on the curated one. As the CDF of duration length of the failure entries depicts more than 40% of the duration lengths are less than 5, implying that failure status is changing relatively rapid. A systematic failure presented at this timestamp might recover at the next timestamp. Hence, in the experiments we test all the methods on both datasets.

In the following sections, performance evaluation of the above methods are carried out on the noncurated and curated datasets. Section 4.4.2 compares methods between using history information and without using history information on the noncurated dataset. Then a method by method analysis as well as a summarized comparison for all six approaches, i.e., SMF, MMMF, MMMFA, SSVD, TENSOR and SMFA, are discussed in Section 4.4.3 and 4.4.4, respectively. In addition, experiments on the curated dataset are also presented. Statistics of the curated dataset is described in Section 4.4.5, followed by a method by method performance analysis on the curated dataset in Section 4.4.6. In Section 4.4.7, we give an overall comparison of all methods on the curated dataset and analyze each method's suitability for dealing with highly imbalanced data. Conclusions of the sequential fault monitoring are drawn in Section 4.5.

For all experiments below we use the first 20 matrices, i.e., $X_1, X_2, ..., X_{20}$, as the initial input set (parameters tunning), and compare each method on the average of 10 times repeats. Concrete parameter settings for these methods are listed in Table 4.3. Parameters marked with a '+' in the table are selected via a train and validation on the first 20 time windows.

4.4.2 MMMF, SMF, MMMFA

We show the performance evaluation on SMF, MMMF and MMMF with active learning (MMMFA) on the noncurated dataset, i.e., a comparison between method with (SMF) and without (MMMF and MMMFA) history information. As presented in Table 4.3, all three methods select 10% of the total entries as training set (i.e., $N = 10\% \times M \times W$). In addition, SMF and SMFA select another 10% of the most confident entries with values from Y_{t-1} according to Algorithm 6. The adaptive weight for these most confident entries is computed according to line 5 in Algorithm 6.

Figure 4-4 illustrates the result. SMF exhibits a significant advantage on six metrics over

Table 4.3: Summary of methods' parameter setting

	Parameters
EWMA	$\theta = 0.5$, damping factor; $L = 20$, input window length; N, 10% of random samples in X_t.
SSVD	N, 10% of random samples in X_t; $R^+ = 10$, rank of SVD approximation; $L = 20$, lag window length for imputation.
MMMF	N, 10% of random samples in X_t; $C^+ = 10$, coefficient for slack penalty; $\Sigma^+ =$*'max norm'*.
SMF	N_r, 5% of random samples in X_t; $C^+ = 10$, slack penalty; N_u, 5% of most uncertain samples in X_t; N_c, 10% of most confident samples from Y_{t-1}.
SMFA	N, 10% of total samples; $C^+ = 10$, slack penalty; $P_0 = 5\%$, initial sample rate; $P_a = 1\%$, active sample rate at each iteration; N_c, 10% of most confident prediction from Y_{t_i}; $\rho^+ = 0.5$, equal size of random samples and most uncertain samples at each active iteration.
TENSOR	N, 10% of random samples in X_t; $R^+ = 10$, # of rank-1 components; λ=ones(10,1), equal weight on each sub-dimension; L=20, lag window length for factorization.
MMMFA	N, 10% of total samples; $C^+ = 10$, slack penalty; $P_0 = 5\%$, initial sample rate; $P_a = 1\%$, active sample rate at each iteration; $\rho^+ = 0.5$, equal size of random samples and most uncertain samples at each active iteration;
A*	$L = 20$, lag window length; $\theta = 0.5$, damping factor.

the other two methods, except that MMMFA reaches a higher performance on sensitivity. Reason for the high sensitivity of MMMFA is as following: positive entries (failures) occupy a much smaller proportion than negative entries (OKs) in X (ground truth), they are therefore more difficult to be detected and are easily fall into the most uncertain prediction set. As a consequence, they are also more likely to be selected out as the most uncertain predictions in the active sampling stage, which in the end results in a high sensitivity of MMMFA.

On the other hand, in addition to most uncertain predictions, SMF also takes the most confident information from the last prediction Y_{t-1}. Given the large proportion of negative entries in X, the majority part of most confident entry set are negative values, which leads to a low FPR for SMF. However, from a balanced measurement view, SMF has the highest

average MCC and FSCORE (Fig. 4-4(e)), which in other words it performs the best in keeping balance between false positive rate and false negative rate.

Similar result can also be observed in the ROC scatter plot. A more concentrated distribution is exhibited by result of SMF in Fig. 4-4(d), showing a lower false alarm rate than the MMMF and MMMFA. Another interesting result is the sharp drop of performance of SMF at the 81st time window (recall that we use 20 time windows as initial input, so this is the 101st time window in the original data). As mentioned before, this is caused by the abrupt changes between observations in two adjacent time windows. Therefore, in this case the history information does not help, but instead impedes the improvement on algorithm's performance. The average result with corresponding standard deviation on five criteria on the test set is presented in Fig. 4-4(e), which from another point of view highlights the superiority of SMF over MMMF. Detailed numeric result of the average performance is illustrated in Table 4.4.

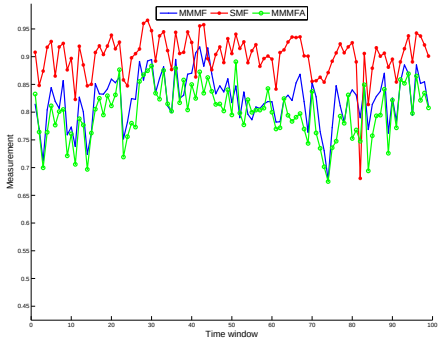
The above analysis emphasizes the effectiveness of using both spatial and temporal information for sequential matrix factorization. On the noncurated dataset, it beats both the simple matrix factorization method (MMMF) and its enhanced version with active sampling strategy (MMMFA). Therefore, a clear conclusion is that, for the noncurated dataset, a proper synthesized use of spatial and temporal information is significantly better than a single usage of spatial information, even with active sampling strategy. Here we only illustrate the algorithms' result without a smoothing on the outputs. However, on the downside, as shown in Section 4.4.4, the smoothed MMMFA* is better than SMF*.

Table 4.4: Average performance comparison for MMMF, SMF, MMMFA

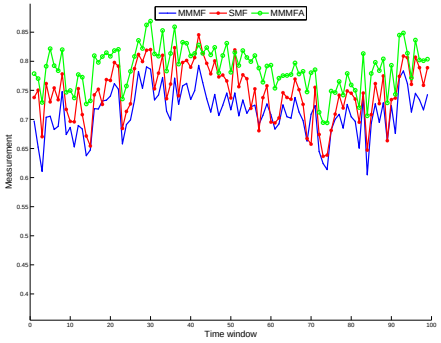
	TPR	SPC	PPV	MCC	FSCORE
MMMF	0.713±0.040	0.970±0.010	0.824±0.045	0.725±0.051	0.764±0.041
SMF	0.747±0.047	**0.985±0.006**	**0.901±0.038**	**0.791±0.046**	**0.816±0.040**
MMMFA	**0.789±0.037**	0.959±0.013	0.800±0.048	0.752±0.052	0.793±0.041

4.4.3 Method by method analysis

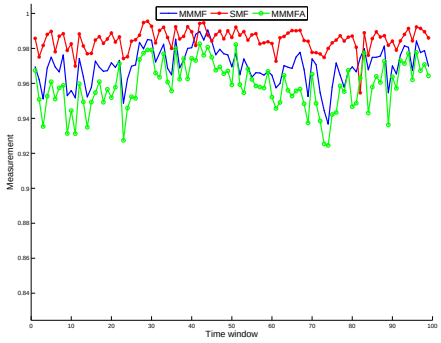
Benefits of using temporal information for the prediction is shown in Section 4.4.2, in this section we present the results of SSVD, MMMF, SMFA and TENSOR on the raw dataset. For each method, EWMA is used as a base line for comparison.



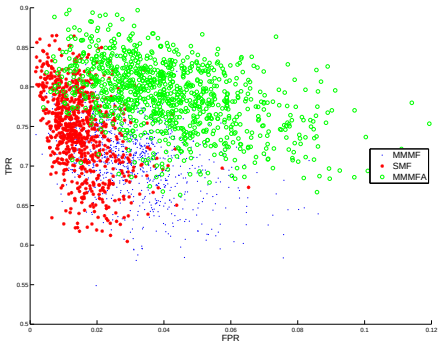
(a) Precision



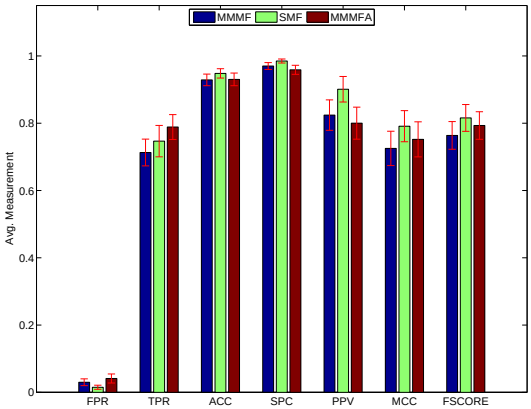
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-4: Performance comparison between MMMF, SMF and MMMFA.

SSVD

Comparison of SSVD, EWMA and SSVD* is presented in Fig. 4-5. As is clearly shown in the precision (Fig. 4-5(a)) and specificity (Fig. 4-5(c)), SSVD (SSVD*) performs significantly better than EWMA. In the average performance plot (Fig. 4-5(e)), the SSVD (SSVD*) exhibits a better performance than EWMA on all the seven measurements except sensitivity. This is reasonable if we have noticed the large FPR value of EWMA. In the ROC scatter plot (Fig. 4-5(d)), we can see that results of EWMA are distributed more sparsely on the right (the upper-left corner $(0, 1)$ is a theoretic optimal point), while SSVD and SSVD* tend to concentrate tightly to the left Y-axis.

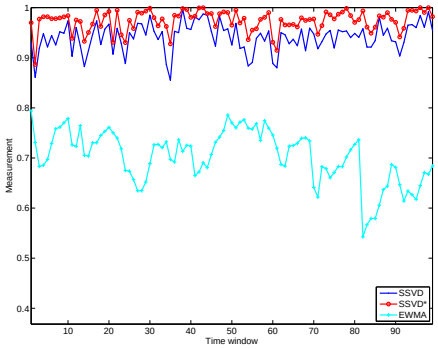
On the other side, there is a visible improvement on the results given by SSVD* to that of SSVD on all measurements, both on individual time (Fig. 4-5) window and on average (Table 4.5), implying that preserving continuity in estimation sequence is beneficial for SSVD. An average performance comparison among these three algorithms is given in Fig. 4-5(e), and its numeric result is shown in Table 4.5.

Table 4.5: Average performance comparison of SSVD with EWMA, SSVD*

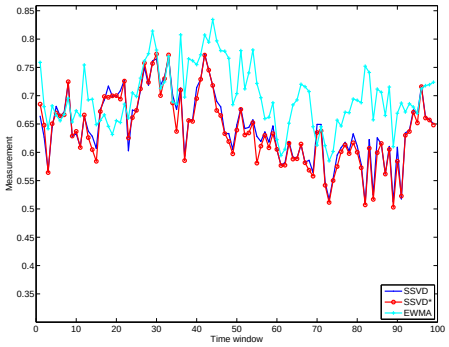
	TPR	SPC	PPV	MCC	FSCORE
SSVD	0.645±0.060	0.992±0.004	0.941±0.028	0.747±0.047	0.763±0.046
EWMA	**0.699±0.054**	0.944±0.011	0.703±0.052	0.643±0.046	0.699±0.037
SSVD*	0.635±0.063	**0.997±0.004**	**0.974±0.034**	**0.757±0.050**	**0.767±0.050**

Discussion

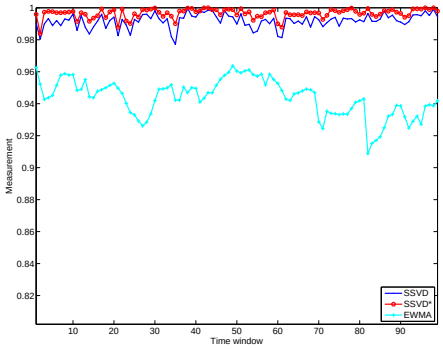
For an imbalanced matrix with negative as the majority, the principal factors preserved after a singular value decomposition mainly reflect the negative population. Those isolated random positive entries are always seen as noise in the SVD and are more likely to be removed after a top rank eigenvector reconstruction. This is the reason why SSVD tends to have a perfect average specificity (0.997 ± 0.004 for the SSVD*) while performs badly on the sensitivity (0.635 ± 0.063). In contrast to SSVD, EWMA tends to favor a positive prediction, which consequently leads to a relative high sensitivity compared with SSVD. Recall that MCC is a balanced measure which considers true and false positives and negatives simultaneously. Therefore, from the balanced point of view, SSVD is superior to EWMA



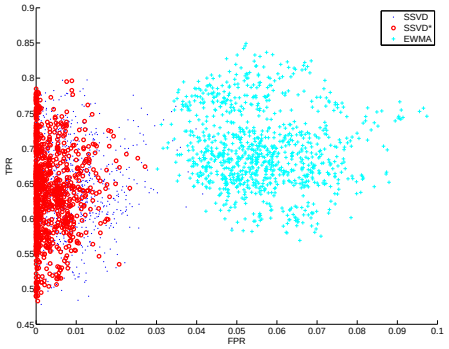
(a) Precision



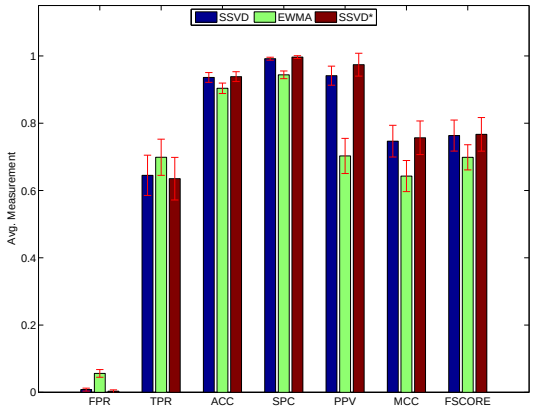
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-5: Performance measurement for SSVD.

since its average value of MCC is clearly better than that of EWMA.

MMMF and MMMF*

Figure 4-6 explains the detailed performance comparison of MMMF, MMMF* and EWMA on individual time windows, where a clear observation is that MMMF* is significantly superior to MMMF and EWMA on precision and specificity. For the sensitivity (TPR), MMMF and MMMF* show similar stable result, while EWMA fluctuates heavily over time. On average, difference in sensitivity is not visible on the three algorithms (Table 4.6).

Table 4.6: Average performance comparison of MMMF, with EWMA, MMMF*

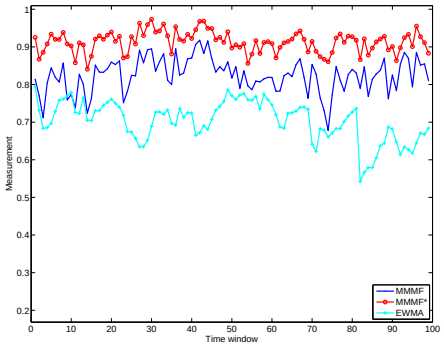
	TPR	SPC	PPV	MCC	FSCORE
MMMF	**0.713±0.040**	0.970±0.010	0.824±0.045	0.725±0.051	0.764±0.041
EWMA	0.699±0.054	0.944±0.011	0.703±0.052	0.643±0.046	0.699±0.037
MMMF*	0.700±0.045	**0.990±0.004**	**0.933±0.031**	**0.778±0.041**	**0.799±0.037**

Discussion

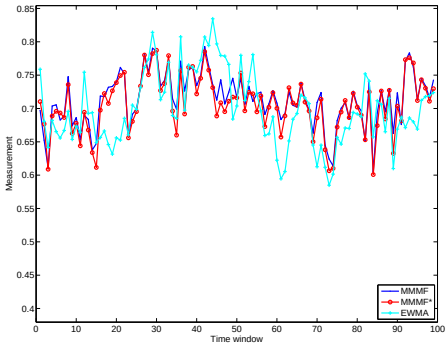
Both MMMF and SSVD can be seen as matrix factorization methods. For SSVD we have seen its tendency of predicting a value (negative in our case) being the majority class of the training set, which results in a low sensitivity and high specificity. For MMMF, as shown in Table 4.6, its sensitivity is better than that of SSVD and EWMA, indicating the MMMF is able to capture the minority positive information during the factorization. Another interesting point is that MMMF is better than MMMF* on sensitivity, but worse on precision, which indicates after a smoothing, the decrease of false positives (higher precision) is accompanied with the increase of false negatives (lower sensitivity). However, the smoothing is still beneficial on average since MMMF* gives palpably better result on the balanced measure MCC and FSCORE.

SMFA

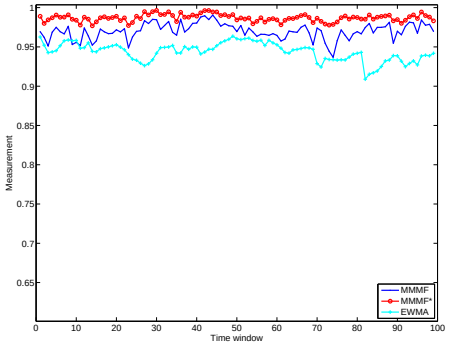
Results of SMFA, EWMA and SMFA* are presented in Fig. 4-7. The performance of SMFA and SMFA* are considerably better than EWMA on all 7 measurements, both on average and on individual time windows. Moreover, SMFA* beats SMFA notably on all measurements but sensitivity, on which the two share similar performance on individual



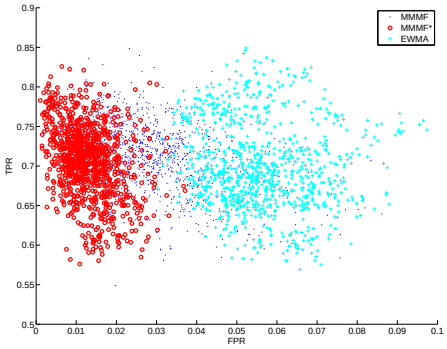
(a) Precision



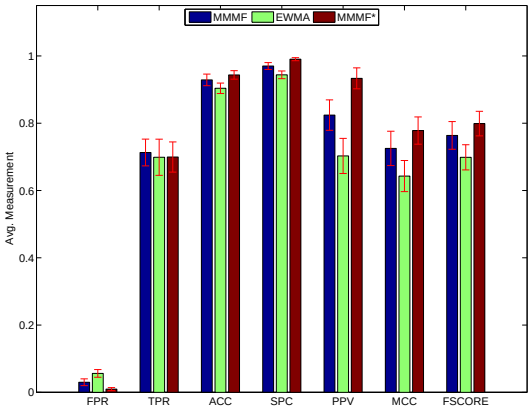
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-6: Performance measurement for MMMF.

time windows and SMFA* is slightly better than SMFA on average. Average performance comparison of the three is shown in Table 4.7 and Fig. 4-7(e). Take the integrated criteria MCC and FSCORE for example, compared with EWMA the improvement of SMFA* on MCC is 34.5% and on FSCORE is 26.5%.

Table 4.7: Average performance comparison SMFA, with EWMA, SMFA*

	TPR	SPC	PPV	MCC	FSCORE
SMFA	0.826±0.047	0.983±0.007	0.907±0.033	0.840±0.046	0.864±0.038
EWMA	0.699±0.054	0.944±0.011	0.703±0.052	0.643±0.046	0.699±0.037
SMFA*	**0.827±0.047**	**0.991±0.005**	**0.950±0.028**	**0.865±0.041**	**0.884±0.036**

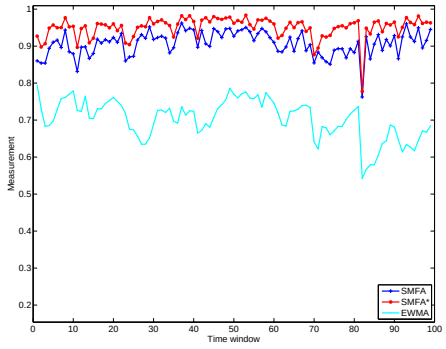
Discussion

The first message delivered in Table 4.7 is the high sensitivity value of SMFA and SMFA*, remarkably better than those of previous mentioned methods. This highlights the advantage of using active sampling as an aid for revealing those *difficult to predict* positive entries. Positive entries are the minority part of the whole population in our case, it is therefore difficult to uncover them by using any conventional method with equal cost on positive and negative entries. However, with the aid of active sampling it is possible to unveil those *difficult to predict* entries, since they are more likely to be exposed and labeled during the active sampling process. Another message lies in the result that SMFA* shows a constantly better performance on all metrics compared with SMFA. In contrast to MMMF* where a decrease in sensitivity is observed after smoothing, SMFA* improves both true positive and true negative rates.

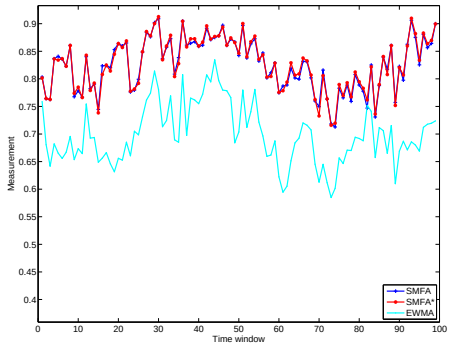
To sum up, sequential matrix factorization with active learning (SMFA) exhibits absolute advantage over single temporal prediction (EWMA), and the smoothing based enhancement (SMFA*) does benefit the raw SMFA.

TENSOR

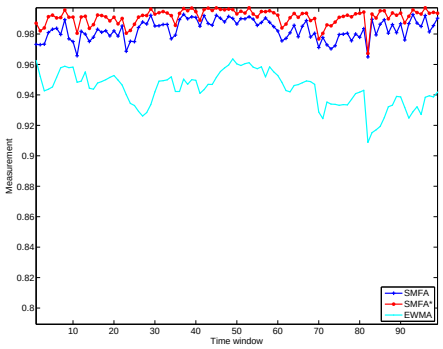
Instead of utilizing information one time window after another, tensor based methods make use of the whole information till the current time window all at once. More precisely, the temporal information that we explored through smoothing or temporal constraint in



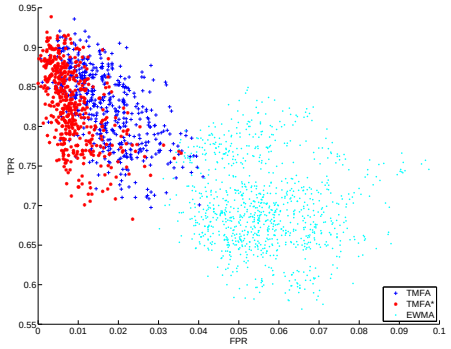
(a) Precision



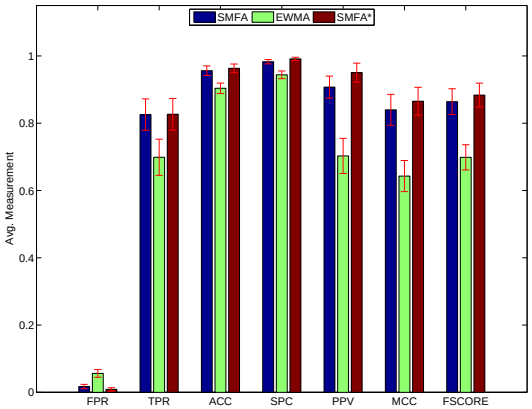
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-7: Performance measurement for SMFA.

the matrix based methods is processed straightforwardly as a separate dimension in tensor based method.

Figure 4-8 illustrates the comparison between EWMA and tensor factorization method. For simplicity, we denote the tensor factorization based method as TENSOR in the following discussion. Parameters setting of this approach can be found in Table 4.3. Similar to the matrix based methods, TENSOR also shows a clear superiority to EWMA on most of the criteria, both on individual time windows and on average (Figure 4-8). However, when comparing sensitivity, TENSOR performs worse than the EWMA, almost on all individual time windows. The poor performance of TENSOR on sensitivity is also shown in the averaged measurements (Table 4.8), where TENSOR is superior to EWMA on all criteria but sensitivity. However the higher sensitivity of EWMA is accompanied with a higher false positive rate. This relatively poor performance of TENSOR is partially due to the fact that in tensor factorization (CAPA) we only perform a single least square estimation for the observed entries, without introducing any regularization.

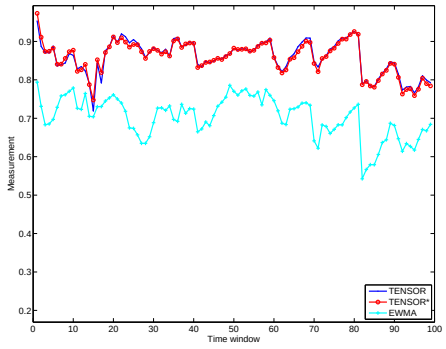
Another interesting result is that, unlike the matrix based methods, discrepancy between results of TENSOR and TENSOR* is not significant. This is clearly exhibited on almost all the criteria (Fig. 4-8 and Table 4.8). One reason may contribute to this phenomenon is TENSOR employs a regression on the time dimension straightforwardly, thus continuity in estimation sequence is already captured, leaving no room for a smoothing based enhancement.

Table 4.8: Average performance comparison TENSOR, with EWMA, TENSOR*

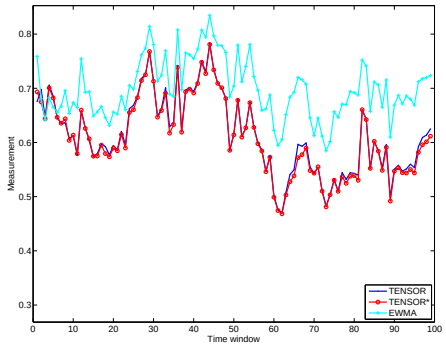
	TPR	SPC	PPV	MCC	FSCORE
TENSOR	0.613±0.071	**0.981±0.006**	**0.859±0.043**	**0.684±0.055**	**0.713±0.053**
EWMA	**0.699±0.054**	0.944±0.011	0.703±0.052	0.643±0.046	0.699±0.037
TENSOR*	0.608±0.072	0.981±0.005	0.856±0.043	0.678±0.057	0.708±0.054

4.4.4 Comparisons between methods on noncurated dataset

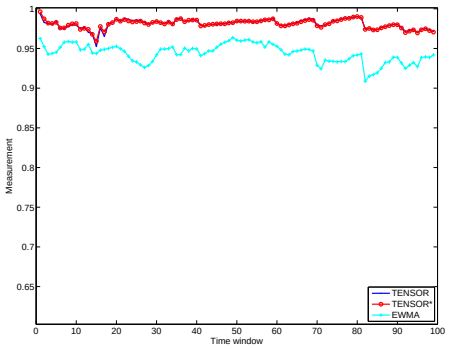
To complete the experiments on the noncurated dataset, we show the comparison between different methods in this section. For each method and its smoothed version the one with better MCC value is presented for comparison. Figure 4-9 illustrates the result of each algorithm on individual time windows, where a few conclusions are drawn as following:



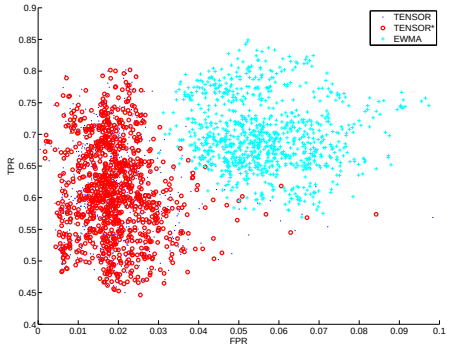
(a) Precision



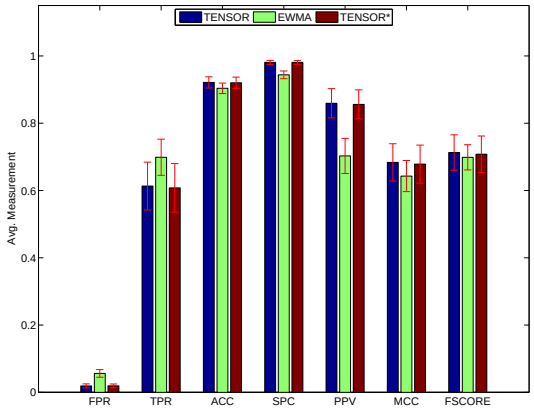
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-8: Performance measurement for tensor.

- SMFA* dominates the other five competitors on three criteria: MCC, FSCORE and sensitivity (Table 4.14). On the other hand, SSVD* dominates all the other algorithms on precision and specificity (Fig. 4-9(c) 4-9(a)), but with a sacrifice of performing badly on sensitivity, MCC and FSCORE, indicating its tendency to favor negative predictions. From a balanced measurement point view, the superiority of SMFA* over other methods on MCC and FSCORE exhibits its ability to control the quality of prediction in terms of balancing true and false positives and negatives.
- The two methods with active sampling, i.e., MMMFA* and SMFA*, occupy the first and second place in the comparison of MCC and FSCORE, respectively, implying that it is always beneficial to introduce active sampling strategy when applicable.
- Except TENSOR, the smoothing trick does help in improving the prediction performance for proposed methods.
- When only compare methods on sensitivity, MCC and FSCORE, their descending order of quality is: SMFA* > MMMFA* > SMF* > MMMF* > SSVD* > TENSOR.

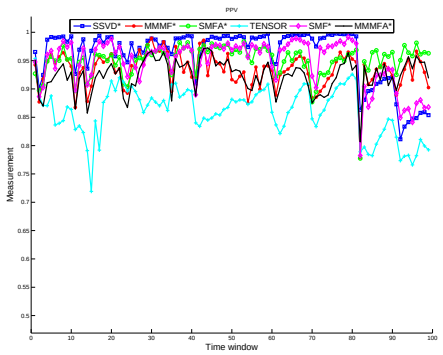
To sum up, in the noncurated dataset, although SMFA* does not give the best (but acceptable) result on specificity and precision, it is still recommended as the best choice for a sequential collaborative prediction task. Under the situation when active sampling is not applicable, e.g., too expensive to launch an active probe, the SMF* would be a good alternative.

Table 4.9: Average performance of SSVD*, MMMF*, SMFA*, TENSOR, SMF*, and MMMFA*

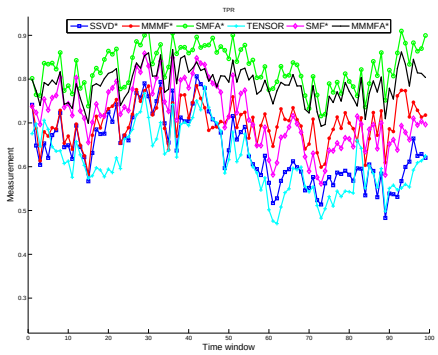
	TPR	SPC	PPV	MCC	FSCORE
SSVD*	0.635±0.063	**0.997±0.004**	**0.974±0.034**	0.757±0.050	0.767±0.050
MMMF*	0.700±0.045	0.990±0.004	0.933±0.031	0.778±0.041	0.799±0.037
SMFA*	**0.827±0.047**	0.991±0.005	0.950±0.028	**0.865±0.041**	**0.884±0.036**
TENSOR	0.613±0.071	0.981±0.006	0.859±0.043	0.684±0.055	0.713±0.053
SMF*	0.716±0.071	0.993±0.005	0.947±0.042	0.797±0.053	0.813±0.051
MMMFA*	0.788±0.039	0.987±0.005	0.924±0.029	0.826±0.038	0.850±0.032

4.4.5 Curated dataset

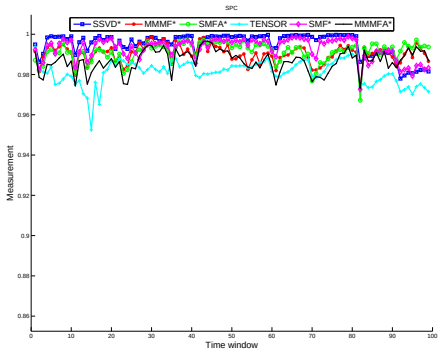
As we have discussed in Chapter 3, it might be more favorable to eliminate the systematic failures in an observed matrix from the reality point of view. Therefore, in this section we evaluate the methods on the curated dataset. Figure 4-10 exhibits the basic statistic of the



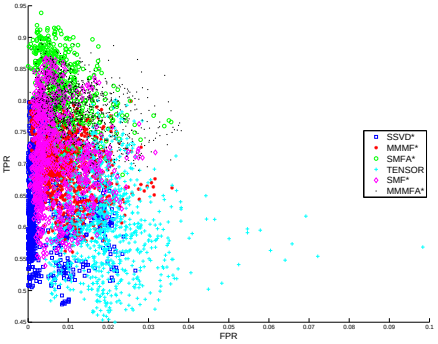
(a) Precision



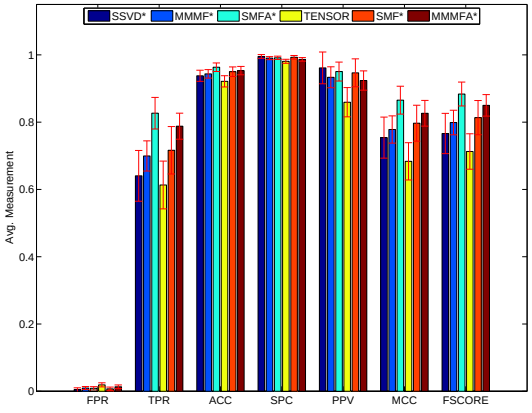
(b) True Positive Rate



(c) Specificity

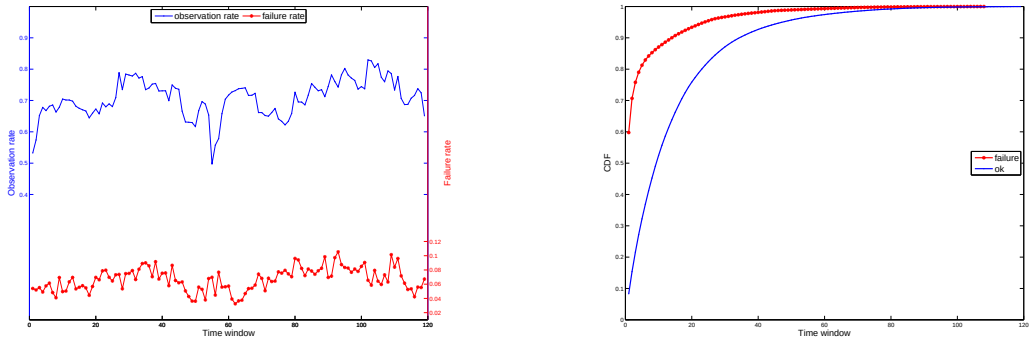


(d) ROC Scatter



(e) Average Performance

Figure 4-9: Performance comparison for SSVD*, MMMF*, SMFA*, TENSOR, SMF* and MMMFA*.



(a) Observation rate and failure rate over time

(b) CDF of duration of OK and Failure

Figure 4-10: Statistics of curated dataset.

curated dataset. Compared with the noncurated data in Fig. 4-3(a), the observation rate of the curated dataset is slightly lower, but a remarkable decrease in the failure rate is clearly presented, approximately from 15% to 5% on average. This means the curated dataset is a highly imbalanced, with the minority holds only 5%. Moreover, the CDF of duration length of failure also experiences a sharp change. The percentage of length-one duration of failure increases from 25% to about 60%, and the percent of failure duration length less than 20 grows from 75% to approximately 92%. In other words, after the elimination of systematic failures, the proportion of short term failure increases, significantly.

4.4.6 Method by method analysis on curated dataset

In this section we show the results of previous methods on the highly imbalanced curated dataset. Parameters setting for the methods tested here are the same as in Table 4.3. Instead of using precision as one of the measurements for comparison in individual time window, in the curated dataset we exhibit the MCC as a replacement, in the aim of comparing methods from a balanced point of view.

SMF, MMMF, MMMFA

Figure 4-11 shows the comparison among SMF, MMMF, MMMFA on individual time windows. Average results are given in Table 4.10. To begin with, recall that in the noncurated dataset the SMF exhibits a clear better performance over MMMF and MMMFA on the criteria of specificity, precision, MCC and FSCORE (Table 4.4), however, on the curated

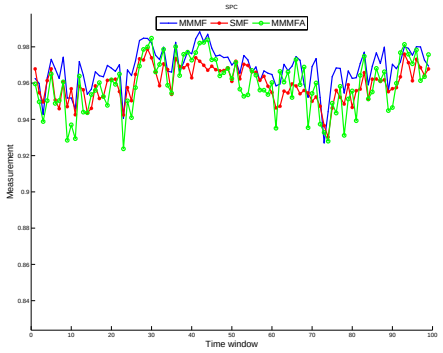
dataset, SMF loses its advantage. Specifically SMF and MMMF give similar results on the averaged MCC and FSCORE, but behave oppositely on specificity (SPC) and sensitivity (TPR). The opposite performance of SMF on the two datasets reveals the difference between the two datasets. In the noncurated case, faults tend to have relatively longer duration, while in the curated case they are more transient. Therefore, in the curated case, heuristic information from the last estimation might not be helpful for discovering the faults in the curated dataset.

Secondly, similar to the noncurated dataset, the sensitivities of MMMFA and MMMFA* are also significantly better than the other two methods and their corresponding smoothed version. In other words, the active sampling strategy is still able to find those positives in the curated case. The superiority obtained by MMMFA on sensitivity, PPV, MCC and FSCORE among the comparison of three non-smoothed methods demonstrates the effectiveness of the active sampling strategy.

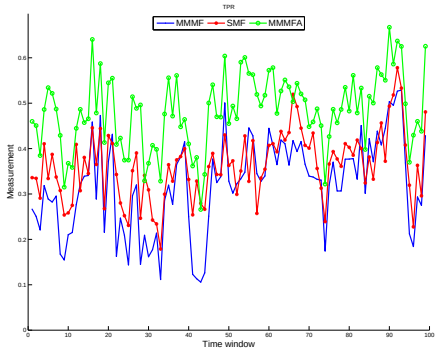
Thirdly, a clear result between the non-smoothed and smoothed version of listed methods is the notable improvement on precision. There is nearly a double growth on precision for each of the method after taking a smoothing on each one's estimation sequence. This is mainly because many false positives in the non-smoothed version are corrected after a smoothing. However, there is a minor cost for this improvement, as shown in Table 4.10, the averaged sensitivity experiences a decrease for each algorithm. From a balanced point of view, it is still beneficial to take a smoothing as a remarkable progress on MCC is clearly exhibited for all three methods.

Table 4.10: Average performance comparison of MMMF, SMF and MMMFA on curated dataset

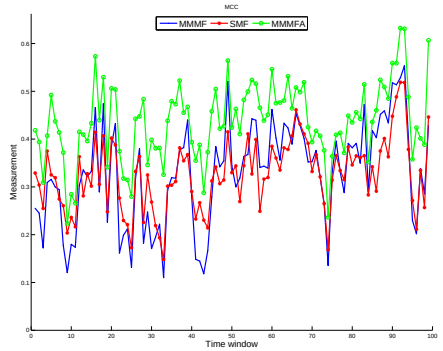
	TPR	SPC	PPV	MCC	FSCORE
MMMF	0.319±0.102	0.968±0.011	0.427±0.110	0.328±0.106	0.361±0.107
SMF	0.362±0.074	0.960±0.009	0.374±0.078	0.326±0.075	0.365±0.074
MMMFA	**0.482±0.081**	0.959±0.014	0.471±0.080	0.436±0.080	0.471±0.077
MMMF*	0.318±0.072	0.995±0.002	**0.832±0.073**	0.493±0.074	0.455±0.082
SMF*	0.306±0.059	**0.996±0.003**	0.821±0.081	0.481±0.058	0.441±0.065
MMMFA*	0.454±0.078	0.988±0.005	0.739±0.071	**0.554±0.073**	**0.558±0.075**



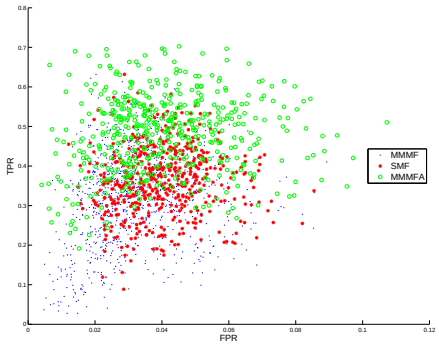
(a) Specificity



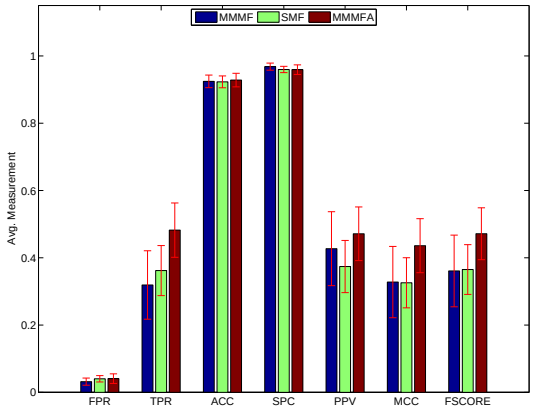
(b) True Positive Rate



(c) MCC



(d) ROC Scatter



(e) Average Performance

Figure 4-11: Performance comparison of MMMF, SMF and MMMFA on curated dataset.

SSVD

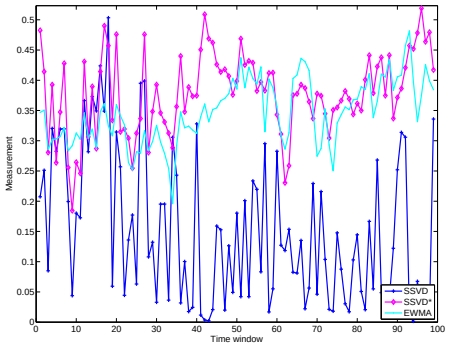
Figure 4-12 compares the performance of EWMA, SSVD and SSVD* on individual time windows on the curated dataset. For the sensitivity, both SSVD and SSVD* fail to obtain a comparable result with EWMA, (with 0.084 and 0.175 for SSVD and SSVD*, respectively) meaning that both SSVD and SSVD* fail to detect most of the positives. However, this is reasonable since in such a highly imbalanced dataset the minority part is only around 5%. Predictions produced by methods (e.g., SSVD) only consider preserving the main factors are unavoidable to be biased by the majority population in the dataset. EWMA, on the other side, procures a relatively high sensitivity but a low precision, indicating there are many false positives concealed in the predictions. Last, SSVD* exhibits constantly better performance than SSVD both on individual time windows and on average (Table 4.11), implying a smoothing action for SSVD is always helpful. To sum up, the low sensitivity of SSVD and SSVD* expose their inability to deal with the highly imbalanced data. Techniques like weighted synthetic oversampling [70] might be a remedy to this.

Table 4.11: Average performance of SSVD on curated dataset

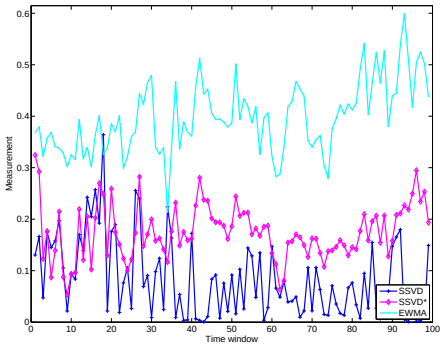
	TPR	SPC	PPV	MCC	FSCORE
EWMA	**0.396±0.069**	0.955±0.007	0.392±0.066	0.347±0.053	**0.389±0.051**
SSVD	0.084±0.076	0.993±0.004	0.398±0.211	0.150±0.124	0.130±0.112
SSVD*	0.175±0.053	**0.999±0.002**	**0.905±0.087**	**0.378±0.067**	0.289±0.074

SMFA

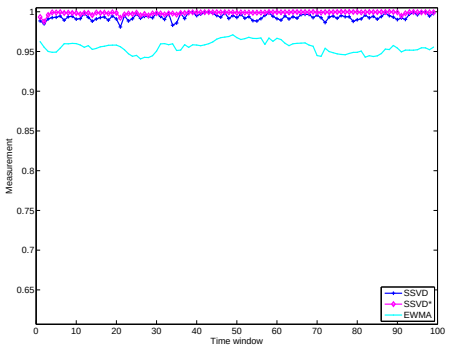
Results for comparison among EWMA, SMFA and SMFA* on individual time windows are given in Fig. 4-13. The first observation is that SMFA and SMFA* show a much better performance over EWMA both on average and on individual time windows. Additionally, SMFA* is constantly better than SMFA on all criteria but sensitivity. As we explained previously, this is due to the corrections made by SMFA* on the result of SMFA are accompanied with a small part of mistakes. Specifically, a clear improved precision and specificity can be observed in the result of SMFA*, which indicates a decrease on the number of false positives. However this improvement is accompanied with a sacrifice on the sensitivity. In other words, after a smoothing, the number of false positives is significantly smaller



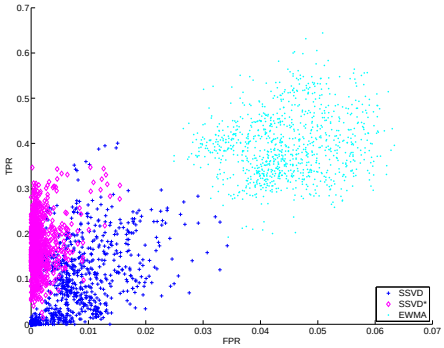
(a) MCC



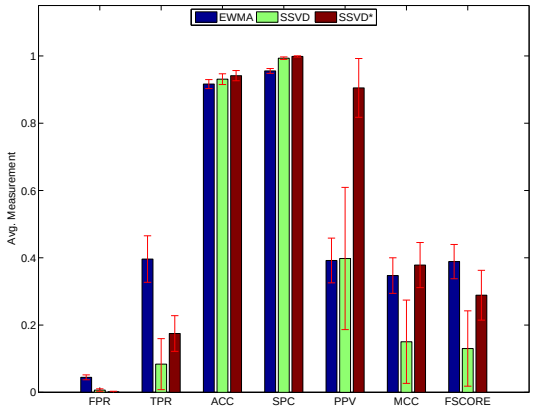
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-12: Performance measurement for SSVD on curated dataset.

(0.993 > 0.986, specificity in Table 4.12) in SMFA* than in SMFA, meanwhile as a cost the number of true positives in SMFA* is decreased compared with SMFA (0.562 < 0.569, TPR in Table 4.12). Nevertheless, this is affordable from the balanced measurement point of view. Because there is an obvious improvement on MCC after exerting a smoothing, both on individual time windows (Fig. 4-13(a)) and on average (Table 4.12).

Table 4.12: Average performance of SMFA on curated dataset

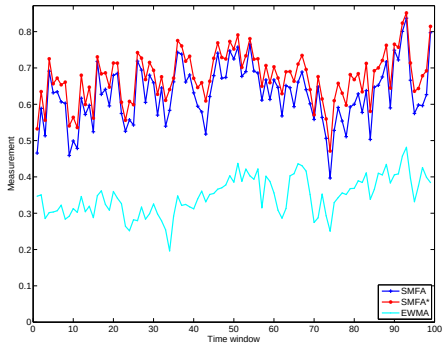
	TPR	SPC	PPV	MCC	FSCORE
EWMA	0.396±0.069	0.955±0.007	0.392±0.066	0.347±0.053	0.389±0.051
SMFA	**0.569±0.076**	0.986±0.006	0.743±0.079	0.628±0.080	0.642±0.076
SMFA*	0.562±0.078	**0.993±0.003**	**0.853±0.053**	**0.675±0.069**	**0.675±0.070**

TENSOR

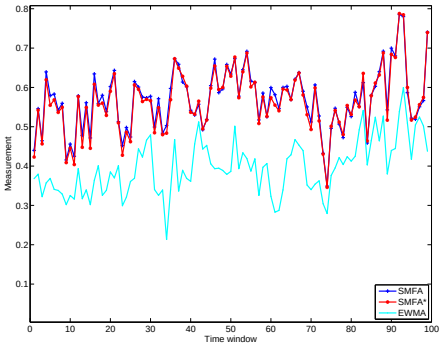
Figure 4-14 illustrates the comparison of different classifier measurements among TENSOR, TENSOR* and EWMA. The TENSOR and TENSOR* demonstrate similar results on the curated dataset. However, similar to SSVD there is a high specificity (Fig. 4-14(c)) but low sensitivity (Fig. 4-14(b)) for TENSOR and TENSOR*, implying the predicted negatives are almost correct while the majority of true positives are miss classified. This is caused by the same reason responsible for the poor performance of SSVD: dimension reduction orientated methods are not appropriate in recovering a partially observed highly imbalanced matrix. From the ROC scatter plot (Fig. 4-14(d)), we can see the difference among the three algorithms: results of tensor based methods illustrate their distribution in the lower-left corner of the figure, indicating a lower FPR as well as a lower sensitivity, while the result of EWMA scatters on the upper right, meaning it obtains a better sensitivity with a cost of higher FPR. On average, the result of TENSOR* has a more concentrated distribution compared with the other two methods.

Table 4.13: Average performance of TENSOR on curated dataset

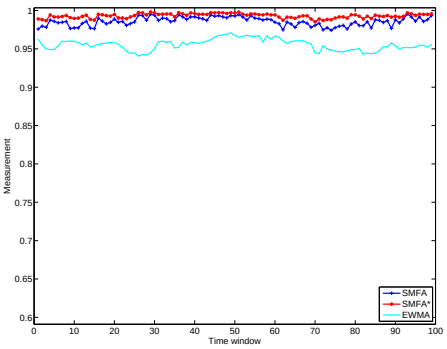
	TPR	SPC	PPV	MCC	FSCORE
EWMA	**0.396±0.069**	0.955±0.007	0.392±0.066	0.347±0.053	**0.389±0.051**
TENSOR	0.275±0.065	0.990±0.003	0.661±0.094	0.397±0.068	0.381±0.069
TENSOR*	0.271±0.068	**0.991±0.004**	**0.692±0.101**	**0.405±0.059**	0.381±0.068



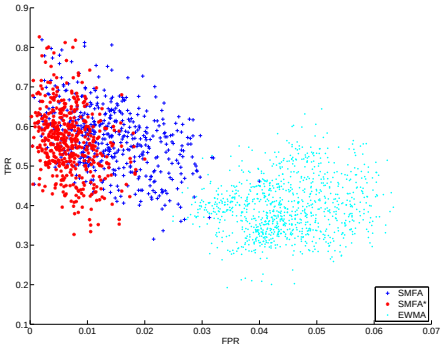
(a) MCC



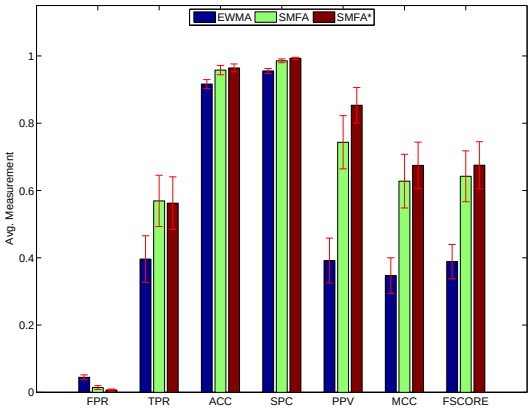
(b) True Positive Rate



(c) Specificity

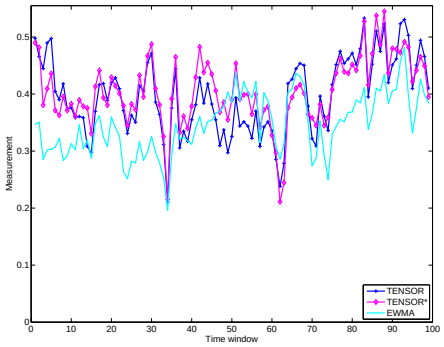


(d) ROC Scatter

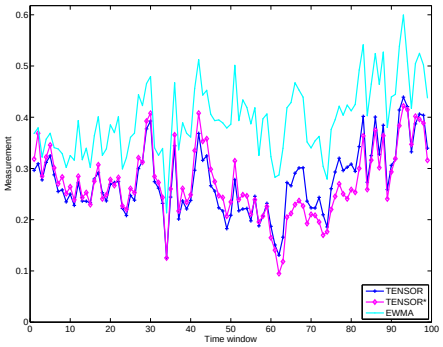


(e) Average Performance

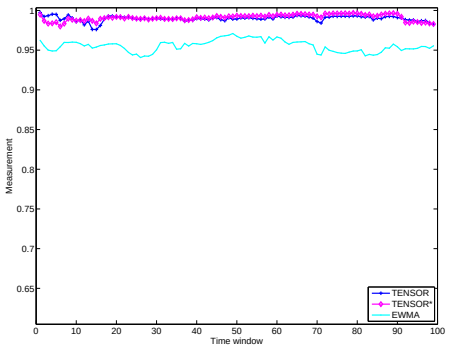
Figure 4-13: Performance measurement for SMFA on curated dataset.



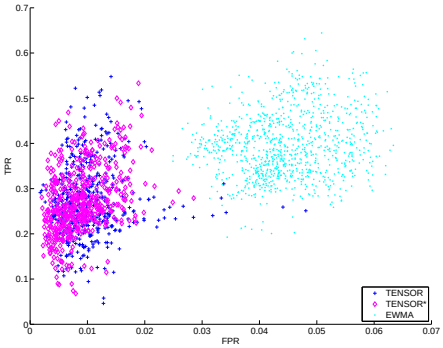
(a) MCC



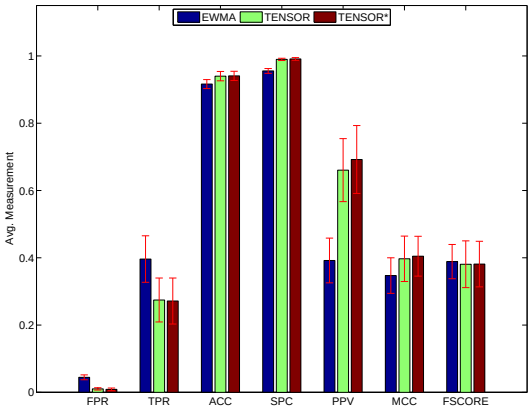
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-14: Performance measurement for TENSOR on curated dataset.

4.4.7 Comparisons between methods on curated dataset

We show the average result of the above algorithms on the curated dataset in Fig. 4-15, corresponding numerical results are listed in Table 4.14. As in the noncurated case, for each method and its corresponding smoothing version we show the one with a better MCC result in the comparison.

Unsurprisingly, on this extreme imbalance dataset all algorithms except SMFA* show a sharp decrease on their performance. For example, none of the algorithms in SSVD*, MMMF*, TENSOR* and SMF* show an average MCC value greater than 0.50, and none of them exhibit a sensitivity better than 0.4. In other words, these methods are unable to discover more than 40% of the faults in the curated situation. On the other hand, methods with active learning like MMMFA* and SMFA* present a drastic advantage over the other algorithms on metrics like MCC and FSCORE, sensitivity and precision. This sharp predomination for MMMFA* and SMFA* instructs us the importance of active sampling while dealing with highly imbalanced dataset.

In detail, compared with other methods SSVD* and TENSOR* show a low average sensitivity (Fig. 4-15(b)) but a high specificity (Fig. 4-15(c)), exposing their weakness in discovering faults in the prediction. As we discussed before this is because that in both methods predictions can be seen as being produced through a single factorization without regularization, which is inappropriate for dealing with highly imbalance dataset.

On the contrary, from Table 4.14 we can see that methods based on max margin matrix factorization (MMMF* and SMF*) obtain a better performance than SSVD* and TENSOR* on discovering the faults (higher sensitivity) and balancing the predictions (higher MCC value). However, the performance is still not good enough in practice since the sensitivity of these two methods is around 30% on average.

Nevertheless, among all the methods MMMFA* and SMFA* procure accomplished performance due to the use of active sampling. Specifically, SMFA* obtains the best performance on sensitivity, MCC and FSCORE, and it's the only method that is capable of discovering more than 50% of the faults while preserving a balanced average MCC value at 0.675 on the curated dataset. Another point is that SMFA* is constantly better than MMMFA* both on average and on individual time windows, suggesting that it is beneficial to make use of temporal information with active sampling in the curated case. If we only

compare methods on sensitivity, MCC and FSCORE, their descending order of quality is: SMFA* > MMMFA* > MMMF* > SMF* > TENSOR > SSVD*.

Active sampling has been shown to be the key to discover the minority class for a prediction task on an imbalanced dataset. The performance obtained by SMFA* is the best compared with the other discussed alternatives, however, in practice further improvement may still needed to meet the application specific requirements, e.g., to discover at least 70% of the faults. In the next section, we try to reform the prediction performance in terms of increasing the active sample size.

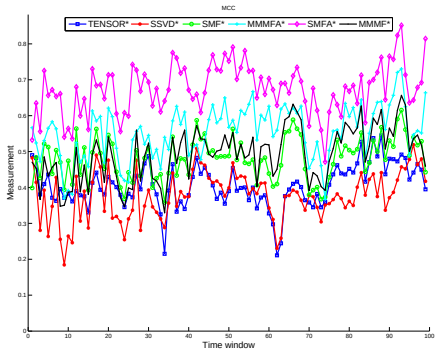
Table 4.14: Average performance of TENSOR*, SSVD*, SMF*, MMMFA*, SMFA* and MMMF* on curated dataset

	TPR	SPC	PPV	MCC	FSCORE
TENSOR*	0.271±0.068	0.991±0.004	0.692±0.101	0.405±0.059	0.381±0.068
SSVD*	0.175±0.053	**0.999±0.002**	**0.905±0.087**	0.378±0.067	0.289±0.074
SMF*	0.306±0.059	0.996±0.003	0.821±0.081	0.481±0.058	0.441±0.065
MMMFA*	0.454±0.078	0.988±0.005	0.739±0.071	0.554±0.073	0.558±0.075
SMFA*	**0.562±0.078**	0.993±0.003	0.853±0.053	**0.675±0.069**	**0.675±0.070**
MMMF*	0.318±0.072	0.995±0.002	0.832±0.073	0.493±0.074	0.455±0.082

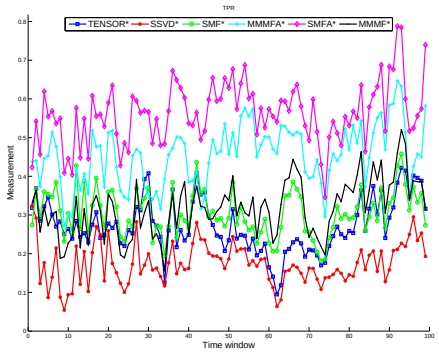
4.4.8 Improving prediction performance on curated dataset

The curated dataset is a highly imbalanced dataset, within which there are only about 5% positive entries. This imbalance causes the classification a real awkward problem to attack and leads to the poor performance of many existed algorithms. Reasons for the ineffectiveness of traditional algorithms on imbalanced datasets lie in the factor that traditional algorithms are accuracy driven, with the aim of minimizing the overall error on which the minority class has little impact. Besides, equal distribution as well as equal cost for the majority and the minority classes are assumed in the algorithms, which of course is not true in real world application. However, among various candidates active learning based methods have their ability to deliver a compelling result. Therefore, in the last part we put our emphasis on the active learning based method and explore a higher training sample rate of SMFA* in the aim of obtaining a more persuasive prediction.

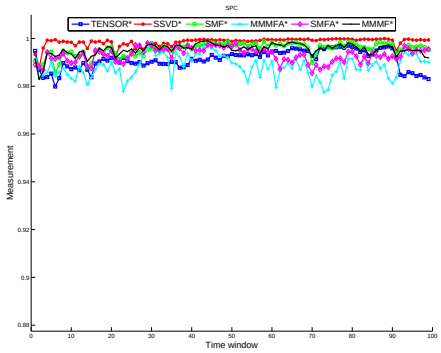
Figure 4-16 illustrates the result of using three sample rates, i.e., 10%, 15% and 20%, of the SMFA* on the curated dataset. Results are averaged on a 5-run experiments. As



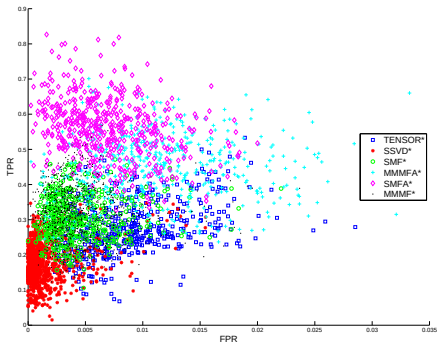
(a) MCC



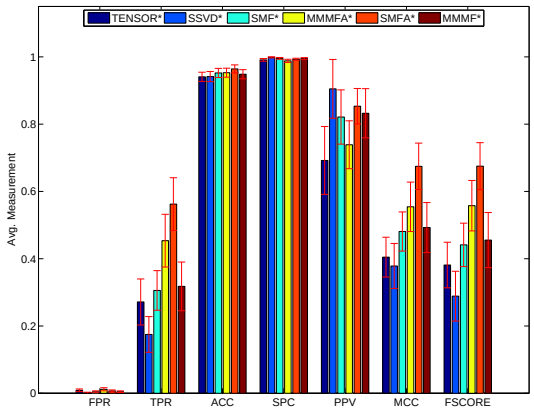
(b) True Positive Rate



(c) Specificity



(d) ROC Scatter



(e) Average Performance

Figure 4-15: Performance comparison for SSVD*, MMMF*, SMFA*, TENSOR, SMF* and MMMFA* on curated dataset.

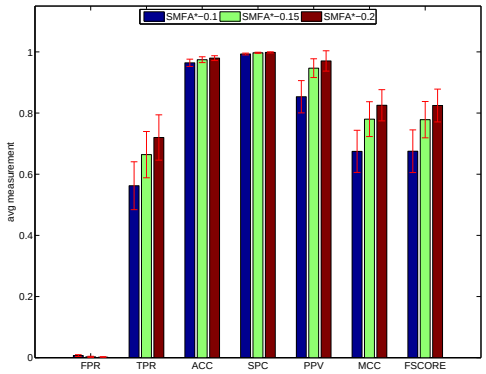


Figure 4-16: Method performance on curated dataset.

illustrated, with the increase of training sample rate in SMFA* a steady and notable improvement is exhibited, together with a significant increase in computational cost. For example, when we take a sample rate of 20%, it is able to find out 72% of the faults while keeping a balanced MCC value of 0.825 on average (Table 4.15). The good news is that it is guaranteed to improve the prediction performance steadily with the increase of sample rate even in the curated situation. However, to control the computational cost within an acceptable level, accelerating methods like Fast MMMF proposed in [132] might be considered.

Table 4.15: Average performance of SMFA* with different sampling rate, curated dataset

	TPR	SPC	PPV	MCC	FSCORE
SMFA*-0.1	0.562±0.078	0.993±0.003	0.853±0.053	0.675±0.069	0.675±0.070
SMFA*-0.15	0.664±0.076	0.997±0.002	0.947±0.031	0.780±0.057	0.778±0.059
SMFA*-0.2	**0.720±0.074**	**0.998±0.002**	**0.970±0.033**	**0.825±0.051**	**0.825±0.054**

4.5 Conclusion

The challenge as well as opportunity brought by sequential data reside in the sequential correlation between consecutive data points. These sequential patterns, as we have shown in this chapter, if exploited properly, can play an important role in improving the prediction performance.

In this chapter, we have introduced several basic methods like EWMA, SSVD, MMMF

and TENSOR to complete the sequential fault prediction task. In order to make full use of both spatial and temporal information concealed in the data, we also propose a sequential matrix factorization method (SMF). Through practice, we have found the effectiveness of using a smoothing action on each method's estimation sequence to improve the prediction performance. Active learning is also employed in combination with the MMMF and SMF such that samples are selected iteratively and actively with the help of inferred heuristic information from the prediction sequence.

To sum up, the following observations can be obtained from the above analysis:

- A smoothing action exerted on the estimation sequence of each algorithm is helpful in improving the overall prediction performance of all methods except that on the noncurated dataset TENSOR exhibits a minor decrease.
- Methods like SSVD and TENSOR are not appropriate in coping with highly imbalanced dataset, e.g., the curated dataset. This is because these methods are designed to minimize the overall error, thus in the highly imbalanced dataset they are unavoidable to favor the majority class and neglect most of the minority class. To deal with the imbalanced dataset, MMMF and SMF do show better performance than SSVD and TENSOR, but are still not applicable on curated dataset. Methods with active sampling strategy procure accomplished results on both datasets, implying the usefulness of active learning in dealing with highly imbalanced dataset.
- An increase in the size of sample set brings an improvement in prediction performance, steadily. However, accelerating methods may be needed to reduce the computational complexity.

Among all the discussed methods, SMFA* shows the best performance on the two mentioned datasets, thus it is recommended as a first choice in this sequential prediction application. Besides algorithms described in this chapter, there are also other alternatives like HMM (Hidden Markov Model) [128], Markov Random Fields (MRF) [91] can be considered. We leave this as a future work to research.

Chapter 5

Sequential Change Detection

5.1 Introduction

The two previous chapters are built on two different assumptions about the temporal behavior of the data. In Chapter 3, time was considered irrelevant (*static* assumption), as all data in a given period were collapsed into one observed matrix. In Chapter 4, the implicit assumption was stationarity, or at least that the past can provide an information about the future. Then, the behavioral model integrates all past (possibly dampened) information. An alternative approach is to consider that ruptures happen, and that the model should be rebuilt at these change points. Because non-stationarity is increasingly recognized as a pervasive phenomenon, change point detection has caught extensive research within diverse areas and application domains. A broad range of real-world applications such as intrusion detection, bio-informatics, system fault management, fraud detection, signal segmentation in data stream are related to the change detection problem. To cite a few, in genomics the goal is to discover changes in gene copy numbers or in the compositional structure of the genome [124, 122]; while in finance, the focus is to detect changes in volatility of time series [6, 10]. For large scale systems, the massive data motivated new data streaming approaches for scaleably and traceably monitoring the system state, involving a self-calibration of the model based on scale invariance [175].

As a consequence, we now consider the *change point detection* problem. This chapter does not propose to add a new algorithm to the enormous body of existing change detection methods, but to define and evaluate a new framework, the *semi-supervised online change detection* (SSOCD). SSOCD integrates offline and online change point detection and exploits

them efficiently.

The chapter is organized as follows. Section 5.2 briefly surveys the change point detection literature in the parts that are relevant for this work. Section 5.3 presents the motivations for our approach in relation with end-to-end monitoring, and formalizes the SSOCD framework. Section 5.4 describes the experimental setting, which is different from the ones in the previous chapters. Section 5.5 presents the results.

5.2 Change point detection

5.2.1 Problem statement

A data sequence can be seen as an ordered series of data with types like binary, discrete, and continuous. Detecting a change in a sequence amounts to identifying the data indexes in the sequence where a sudden change of its properties, e.g., mean, variance, or both, takes place. A more general statistical formulation of change point detection is given by considering the probability distribution from which data before and after a target time point are generated.

Let $X = \{x_1, x_2, ..., x_n\}$ denote the series of observations, and $\tau \geq 0$ be a deterministic unknown change point (the minimax formulation, see [125]), such that for $X_1^\tau = \{x_1, x_2, ..., x_\tau\}$ follow a distribution $\mathbb{P}_\infty$ with density f_∞ and $X_{\tau+1}^n = \{x_{\tau+1}, x_{\tau+2}, ..., x_n\}$ follow another distribution $\mathbb{P}_0$ with density f_0. For $\tau = 0$, it means there is a change at the very beginning of the sequence, and for $\tau = \infty$, it means no change is present in the sequence. In addition, for $d = \{0, \infty\}$, let $f_d^{(n)}(\cdot)$ be the density of probability measures $\mathbb{P}_d^n$. The density of a change point τ, $p_\tau(X)$, is given as following:

$$p_\tau(X) = (\prod_{j=1}^{\tau} f_\infty^{(j)}(X_j|X_1^{j-1})) \times (\prod_{j=\tau+1}^{n} f_0^{(j)}(X_j|X_1^{j-1})), \tag{5.1}$$

where $f_\infty^{(j)}(X_j|X_1^{j-1})$ and $f_0^{(j)}(X_j|X_1^{j-1})$ are the conditional densities of the j-th observation, X_j, given past observation sequence X_1^{j-1}, $j \geq 1$. Equation 5.1 is a very general formulation, without any assumption neither on independence nor on homogeneity of the data, i.e., the observations can be arbitrary dependent and non-identically distributed [125]. This formulation can be concreted into a more prevalent form where observations are assumed

to be independently and identically distributed (i.i.d):

$$p_\tau(X) = (\prod_{j=1}^{\tau} f_0(X_j)) \times (\prod_{j=\tau+1}^{n} f_1(X_j)), \tag{5.2}$$

where $f_0(x)$ and $f_1(x)$, $f_0(x) \neq f_1(x)$, are two density functions of observations before and after the change point τ. In the following paper, without particular explanation, we will only focus on the i.i.d. case.

Based on the way the observed data being processed, detection approaches can be divided into two classes: offline (batch) and online (sequential) change detection.

- Offline change detection (batch): In this case, the task is to find whether or not one or several change points exist in a sequence of n observations, $X = \{x_1, ..., x_n\}$. All available information, i.e., all the n observations, are used to determine whether a change has occurred at a specific point in X.
- Online change detection (sequential): In this case, the length of the sequence is not fixed, but instead, is growing with new arriving observations over time. In contrast to batch methods, the online change detection algorithm makes a decision each time a new observation arrived only based on observations received so far. Observations are processed in a sequential way until a change is detected - the algorithm *signals*, which in a monitoring context entails raising an alarm. Then the detection algorithm will restart from the next observation in the sequence.

Generally, a change detection method is shaped by the employed techniques, e.g., statistical test, information criterion or optimization method used. Based on the extent to which the changes are detected, we categorize the change detection methods into three classes: statistical test based methods, optimization based methods and learning based methods. Table 5.1 illustrates the topology (just another word for categorization) of this classification and a few representative methods in each class. We do not show a concrete example in the class of optimization based methods for online change detection, as we consider most of the related works fall into the other two categories. Any way, this categorization is not a precise description of the enormous change detection filed, but instead one of many attempts to characterize different approaches from various points of view. In the following, we take a brief survey on several representative methods in each category.

Table 5.1: Categorization of change detection methods and representative examples

	Statistical test based methods	Learning based methods	Optimization based methods
Offline	Energy statistic [139, 82]; F-test, Levene's test, Bartlett's test; Student t-test, Mann-Whitney test.	Bayesian models [20, 27, 47, 48, 168];	Pruned Exact Linear Time method [90]; Total Variation Denoising [105]; Segment Neighborhood [16]; Optimal Partitioning [170, 81]; Minimum Description Length [43].
Online	Shewhart control chart; CUSUM; Page-Hinkley test; Generalized likelihood ratio test.	Bayesian online methods [4, 49]; Kernel change detection [44]; Density-ratio estimation [86]; Switch distribution [163]; Singular Spectral Analysis [121].	

5.2.2 Topology of offline methods

Statistical test based methods

Let $D_{k,n}$ denote the two sample test statistic for the sequence $X = \{x_1, x_2, ..., x_n\}$ at time k $(1 \leq k \leq n)$, $h_{k,n}$ denote the decision threshold. Since the location of a change point is unknown in advance, $D_{k,n}$ is always computed over all possible k, for $1 \leq k \leq n$. Under the assumption that there is at most one change point in X, the test statistic D_n is computed as following [140]:

$$D_n = |\max_k \frac{D_{k,n} - \mu_{D_{k,n}}}{\sigma_{D_{k,n}}}| \quad 1 \leq k \leq n, \tag{5.3}$$

where $\mu_{D_{k,n}}$ and $\sigma_{D_{k,n}}$ are the mean and standard deviation of $D_{k,n}$, respectively.

The null hypothesis that no change in presented in X can be rejected if $D_n \geq h_n$ for some threshold h_n. h_n is selected such that the overall false alarm rate α (type-I error) is bounded. However, due to the lack of an analytic finite-sample expression of $D_{k,n}$, it is generally hard to define an exact threshold $h_{k,n}$ that meets the type-I error α. For some specific test statistics $D_{k,n}$, its asymptotic distribution of D_n can be derived as a replacement. For instance, [123] gave the distribution for the Mann-Whitney test, [69] derived the distribution when $D_{k,n}$ is associated with the Student-t test, and in [167] a general procedure for computing asymptotically bounding D_n of other classes of test statistics is derived. The problem of these asymptotic distribution lies in the inaccuracy when the length of X is finite. To ease this, numeric simulations such as MCMC can be employed to estimate the distribution [140].

Change detection methods based on conventional statistical tests generally rely on the

assumption of the distribution of the data, e.g., normality. To overcome this, James et. proposed a hierarchical nonparametric method based on the energy statistic [139] for change analysis in [82]. The only assumption on data is, for some $\alpha \in (0, 2]$, the absolute αth moment exists, and data are independent over time. It is capable of detecting any type of distributional change like change in mean, variance, tail. We will give a detailed introduction of this method in Section 5.3.3.

Optimization based methods

Another line of research of offline change detection directs to identify the locations of multiple change points based various optimization techniques, e.g., dynamic programming, genetic optimization. A general considered objective function is to minimize a loss function within each split segment, coupled with an information criterion for avoiding over fitting (Eqn. 5.4).

$$\sum_{i=1}^{m+1} [C(X_{\tau_{i-1}+1}^{\tau_i})] + \beta f(m), \tag{5.4}$$

where C is a segment based cost function and $\beta f(m)$ is a penalty to avoid over fitting. A commonly used cost function C within each segment is twice the negative log likelihood [75, 33]. Other cost functions such as quadratic loss [135], cumulative sums [79] and combined loss based on both the segment log-likelihood and segment length [174] are also proposed for alternative selection.

The penalty function $\beta f(m)$ for controlling the number of change points also exists several choices. The most common one in practice is a linear penalty with the number of change points, i.e., $\beta f(m) = \beta m$. AIC [8], ($\beta = 2p$), BIC [148] (also known as SIC, $\beta = p \log n$) are typical examples of this linear penalty (Here p is the number of additional parameters brought by adding a new change point). In addition, difference between consecutive segments is also used to penalize the selection of change points [105]. Other choices other than these penalties are discussed in [67, 124, 24].

Typical methods like Segment Neighborhood (SN), Optimal partitioning (OP), Pruned exact linear time method (PELT), Total variation denoising (TVD, see Section 5.3.3.) fall into this category.

SN is a method proposed by [16] for multiple change points detection. Its essence is to define a measure of data fit and penalty function as in Eqn. 5.4, and use a dynamic programming to search the entire sequence. It begins by setting a maximum number of change points, Q, and then computes the cost function for all possible segments, i.e., segments with between 0 and Q change points. The computational cost of this method is $O(Qn^2)$ because of the exhaustive search for all possible segments.

An Optimal partitioning approach (OP) which aims at minimizing Eqn. 5.4 with $f(m) = m$ is proposed in [170] and [81]. A dynamic programming is employed for an iterative optimization of the objective function. According to [81] the OP method is able to automatically determine the number of change points and is guaranteed to find the exact global optimum with a time complexity of $O(n^2)$.

Moreover, to make the computation of OP even more efficient, a modified method with $O(n)$ complexity, denoted as PELT (pruned exact linear time method, see Section 5.3.3), is proposed in [90]. The key improvement in PELT is to discard those candidate change points which can never be a minimum of the objective function in OP.

Other than Eqn. 5.4 objective function based on the minimum description length (MDL) is also considered for offline segmentation. The MDL principle [65] is a formalization of Occam's Razor, which selects the best model for a given set of data as the one with the best compression of the data. To find the best set of change points under the MDL criterion, genetic algorithm is used for searching the objective in [43].

Learning based methods

Besides the above mentioned methods, algorithms based on statistical learning also emerged for offline segmentation. [27] described a Bayesian method that identifies DNA segments using a hidden Markov chain model. [20] proposed a product partition model (PPM) which assumes observations as well as probability of a change occurring at an observation are independent. The observation sequence can be split into an unknown number of K segments, with each admitting an Gaussian distribution independently. Under the assumption that different segments are non-overlapping, Fearnhead developed an efficient dynamic programming methods for the exact computation of the posterior over the number and location of change points in time series [47, 48]. Xuan and Murphy [168] extended Fearnhead's approach to multidimensional time series by introducing sparse Gaussian graphical models to

shape the correlation structure of the vector-valued observations in high dimensions.

In the Bayesian model, numerical simulations such as MCMC and reversible jump MCMC are always applied to generate samples from the posterior distribution, such that change points and their positions as well as the segment-wise parameters can be estimated. However, this simulation process is always computational expensive and is hard to detect its convergence, thus efforts have been put for the improving the simulation efficiency. [45] proposed a two-stage forward-backward algorithm for hidden Markov models in this aim, where a forward pass through the data is conducted in the first stage, following by a simulation of change points backwards in time. Similar work can be found in [19, 107].

5.2.3 Topology of online methods

In this section, we will discuss the online change detection problem under the i.i.d. assumption. Particularly, methods are classified into two categories: methods based on statistical test (e.g., CUSUM, Shewhart control charts), and methods based on learning techniques (e.g., Bayesian online change detection, kernel change detection).

Statistical test based methods

Alike the offline case, the likelihood ratio can also be used for the online change point detection. Formally, let Γ be the change point variable, the observation sequence X has probability mass function (p.m.f.) f_0 before the change point, and f_1 after the change point. The log-likelihood ratio for observations X_j^k (observations from x_j to x_k) can be defined as:

$$S_j^k = \sum_{i=j}^{k} s(x_i) = \sum_{i=j}^{k} \ln \frac{f_1(x_i)}{f_0(x_i)}, \tag{5.5}$$

where $s(x_i)$ is called the sufficient statistic [21] with expected positive value under f_1 and negative value under f_0. The sign of the mean log-likelihood ratio is used to indicate the change of the parameters. According to the Neyman-Pearson lemma, the optimal decision rule d is given as follows:

$$d = \begin{cases} 0 \;\; if \; S_1^N < h; \quad choosing \;\; H_0 \\ 1 \;\; otherwise; \quad choosing \;\; H_1, \end{cases} \tag{5.6}$$

where h is pre-defined a threshold.

Many conventional methods are based on an extension of the likelihood ratio test, which basically can further be divided into two classes: pre-change and post-change distributions are known, and pre-change distribution is known while post-change distribution is unknown. Methods like the Shewhart control chart, geometric moving average (GMA), CUSUM and Page-Hinkley test are representative examples in the first class. On the other hand, weighted CUSUM and the generalized likelihood ratio test (GLR) are two commonly used techniques in the second class [21].

Learning based methods

As introduced in Section 5.2.2, a wide range of retrospective Bayesian approaches have been proposed to the offline change point detection problem. Here we give a brief review of typical learning based online change detection methods .

[4] proposed an online approach based on Bayesian analysis, whose key concept is the time since the last change point, i.e., *run length*. An online prediction for the run length is conducted at every observation, given an underlying predictive model (UPM) and a hazard function learned from past observation. More details of this Bayesian online change detection will be given in Section 5.3.4 under the SSOCD framework. In [49], an online algorithm for exact filtering of multiple change points is introduced, in which the simulation for the number and location of the change points are performed from their corresponding true joint posterior distribution. In addition to the Bayesian models, [44] presented an online *kernel change detection* algorithm, in which the dissimilarity between consecutive subsequences is measured on their feature space using a soft margin single class SVM. [163] identifies the catch-up phenomenon as a new explanation for the slow convergence of Bayesian methods, and proposes a modification of Bayesian predictive distribution, named as the switch distribution, for the task of result prediction, model selection and adaptive estimation. This switch distribution is statistically consistent under fairly weak conditions, and adaptive estimation based on it has shown to be optimal relative to the cumulative Kullback-Leibler risk in a general i.i.d. data setting.

Another line of this category directs to subspace analysis. [121] performs the change detection by first using a sequential application of singular-spectrum analysis (SSA) to the sub-series of original data to learn a basic subspace, and then tracking the quality

of the approximation of other parts of the data series on the learned subspace. In [87] a geometric method is used to estimate the linear state-space model (SSM) of time-series data. Change point detection is achieved in terms of estimating the column space of the extended observability matrix of the learned SSM, and evaluating the subsequence of new-arrived data based on this subspace.

Besides the above model based methods, a number of general model-free methods have also been proposed for change detection. [86] proposed a non-parametric method using a direct density-ratio estimation which does not rely on a strong model assumption. Other typical examples are time-frequency approaches [99], wavelet approaches [40, 72], spectral coefficients approaches [149].

5.2.4 Performance criteria

Theoretical criteria

The classical performance criteria for evaluating the theoretical performance of change point detection algorithms are [21]:

- probability of false detection
- probability of nondetection
- accuracy of the change time and magnitude estimates
- mean delay for detection
- mean time between false alarms

As we will show below, they are ill-suited to empirical validation. We briefly discuss them nonetheless.

The first three criteria are related to off-line change detection. They address two questions: *whether* there is change in the sequence and *where* (change time estimation). The first two criteria (*probability of false detection* and *probability of nondetection*) simply rephrase the usual power and size criteria of hypothesis testing, or sensitivity and specificity in classification vocabulary. The accuracy and magnitude criteria derive from the probability distribution of the change time estimation error, e.g., $P(\hat{t}_0 = t_0 \pm n)$ or $P(|\hat{t}_0 - t_0| \leq n)$, with t_0 be the true change point location and $\hat{t}_0$ be its estimation.

In the online change detection problem, the *mean delay of detection* and *mean time between false alarms* enrich the sensitivity/specificity indicators with timeliness-related char-

acteristics. Generally, the objective is to minimize the delay of detection for a fixed mean time between false alarms. A composite measure for investigating the properties of online change detection algorithms called *Average Run Length* (ARL) was proposed in [14], with a focus on the expected interval between false positives. Intuitively, a lower value of the ARL leads to a faster change detection, at the sacrifice of higher false positives. In the theoretical case, computing closed forms for the ARL of a detection method is generally impossible and replaced by Monte Carlo estimation (see e.g., [13] for a recent survey).

Empirical criteria

The above mentioned criteria are well-suited to a theoretical context, where validation assumes the availability of ground truth in the form of probability distributions for the data, or at least knowledge of the change points. In the fault monitoring context, properly labelled dataset do not exist (see Section 5.3.1). Then, the performance of a change detection algorithm must be evaluated only from its own output. We briefly describe here two possible approaches.

The first one considers the general context of distribution equality. Specifically here, the question is whether two consecutive segments have the same distribution. As there is no reason to assume normality (t-test), the classical indicator is the Kolmogrov-Smirmov statistic. For the sake of completeness, we briefly recall its definition and properties.

Kolmogrov-Smirmov statistic and test. The two-sample Kolmogorov-Smirmov statistic [115] is a nonparametric and distribution-free statistic that measures the difference between the empirical distribution function of two data samples. It has the advantage of making no assumption on the distributions. The statistic is calculated under the null hypothesis (that samples are from the sample distribution) as following:

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)|, \tag{5.7}$$

where $F_{1,n}$ and $F_{2,m}$ are the empirical distribution functions of the first and second sample respectively.

Many other distance measures can be considered, in particular, from an information theoretic approach, those related to mutual information, e.g., Kullback-Leibler or Jensen-Shannon divergences. However, they are less easily amenable to testing.

The second approach considers segmentation as a clustering, where the clusters are the segments. Without label, the rand index is a frequently used measure in clustering which evaluates the agreement between two segmentations of the same data.

Rand Index [129]. Suppose a sample set X with T observations are given by two partitions $U = \{U_1, ..., U_a\}$ and $V = \{V_1, ..., V_b\}$, with a and b segments, respectively. For these two partitions, two variables are defined:

- $\#A$, number of pairs of observations in X that are in the same set in U and in the same set in V;
- $\#B$, number of pairs of observations in X that are in different sets in U and in different sets in V,

then the Rand Index is calculated as following:

$$R = \frac{\#A + \#B}{\binom{T}{2}}. \tag{5.8}$$

Intuitively, $\#A+\#B$ can be seen as the number of agreements between U and V. One simple example, suppose $X = \{x_1, x_2, x_3, x_4, x_5\}$, and the two partitions $U = \{< x_1, x_2, x_3 >, < x_4, x_5 >\}$, $V = \{< x_1, x_2 >, < x_3, x_4, x_5 >\}$, then according to the definitions given above $\#A = 2$, $\#B = 4$, and the rand index is $R = \frac{2+4}{\binom{2}{5}} = 0.6$.

However, one disadvantage of the Rand Index is that it is not suitable for comparing two different estimated segmentations because of its inability to evaluate the discrepancy from a given baseline model. As pointed out in [77], the Rand Index does not take on a constant value when comparing two random clusterings (The expected value of the rand index of two random partitions does not take a constant value, e.g., 0.). Therefore, in [77] and [55] a hyper-geometric model of randomness which conditions on both the number of clusters and their sizes is introduced. An Adjusted Rand Index is then defined as:

$$AR = \frac{R - E(R)}{1 - E(R)}, \tag{5.9}$$

where $E(R)$ is the expected Rand Index and 1 corresponds to the maximum Rand Index value. This adjusted index has expected value zero for independent clusterings and maximum value one for identical clusterings. The adjusted Rand Index value of the above example is: $AR = \frac{0.6-0.52}{1-0.52} = 0.1667$.

5.3 Semi-supervised online change detection

This section first motivates SSOCD (section 5.3.1) , then defines and formalizes it (section 5.3.2). The next sections are devoted to a detailed descriptions of its components.

5.3.1 Motivation

The major problem of online change detection methods is their inability to look ahead, lacking a global view compared to their offline (retrospective) counterparts. However, in the monitoring context, online detection is obviously required. Another major application of online monitoring is data streaming, when the amount of data is too large to be processed in as a whole; although this is not the main focus here, algorithmic scalability has to be assessed in this perspective.

We first summarize the configuration of online change point detection for fault monitoring, then detail each of its points.

- Completely non-supervised online change point detection is likely to be ineffective in a noisy environment.
- Ground truth is not available.
- Relatively reliable labels can be obtained.
- Finally, we propose to use the better quality labels to overcome the limitations of the non-supervised approach.

In Section 5.2.3 we have introduced the current work on the unsupervised online change point detection, where methods depend either on detecting changes in statistical properties of the data or on the analysis of subspaces where data sequence are constrained [78, 121, 87]. Although these methods provide extensive means for change detection, the completely unsupervised manner which works as a black box might bring undesirable results in detection.

Firstly, all sorts of changing behavior in the data sequence might be altered as change points regardless of their irrelevance. Specifically, various types of changes like transient change, oscillating change, periodical change and long term evolving trend may present simultaneously in the data sequence. However, despite the diversity of changes only a few of them are of real interest. Thus online algorithm which is capable of detecting the intended changes as the offline alternatives becomes increasingly important in real applications. Moreover, noise, outliers as well as contamination are common ingredients of real

world data. These unimportant but harmful parts, if not properly processed, may result in the ineffectiveness of online methods.

To make situation worse, in the online change detection setting a careful preprocess of these undesired data is always unavailable due to the time demand of online detection application. Thus, if a detection method is not robust enough, one simple expectable consequence is a high false alarm rate of it brought by these small "salts". Lastly, methods like CUSUM and Page-Hinkley test heavily rely on a predefined threshold for change detection, which makes the tradeoff balancing between false alarm rate and detection delay a complex task in terms of tuning the threshold.

Nevertheless, a supervision in training the detection model when labels are available would yield better performance in detecting desired changes. One could train the algorithm to detect only those types of wanted changing behaviors by feeding change labels. Practically, in many applications a set of well labeled change points may be correlated to desirable events which are of vital application specific interests, thus a change point model trained on labeled data set might be more appropriate for detecting desired changes.

On the downside, one factor that hinders the spread of supervised change detection method lies in the difficulty of acquiring labeled change points. The acquisition of labeled change points from a large data sequence often requires a skilled human expert (e.g., audio segmentation) or a physical experiment (e.g., determining the 3D structure of a protein). In the monitoring case, ground truth would require to actually know what was the cause of the fault; we explained in the previous chapters why automating this discovery is not realistic. The human experts would be the system operators, notoriously difficult to enroll in a non-operational program. Overall, this process of labeling is always expensive, difficult or time consuming which renders a fully labeled training set infeasible.

Meanwhile, unlabeled offline data are always easy to obtain, but are only processed by offline segmentation algorithms. Offline methods described in Section 5.2.2 like the segment neighborhood (SN) provide plenty choices for dealing with data series. Since a global view of the data set is available, a more reliable exact partition is easy to obtain using offline segmentation with desired features. With the assistance of offline segmentation methods, relatively reliable change point labels become available.

Based on the above analysis, we propose a semi-supervised online change detection framework to facilitate the online detection of desired changes in the next section. Different

to conventional semi-supervised learning methods where typically a small amount of labeled data and a large amount of unlabeled data are available, only unlabeled data is available in our situation. "Virtual" labels obtained by unsupervised offline methods are used to train supervised online detection methods. Since our work is an attempt to combine the unsupervised offline and the supervised online leaning methods, we name the framework as a semi-supervised one. In the following, we first propose the framework in Section 5.3.2, and then introduce possible choices for the unsupervised offline and supervised online change detection methods in Section 5.3.3 and 5.3.4, respectively.

5.3.2 Semi-Supervised online change detection framework, SSOCD

Our main contribution in this chapter is to propose a semi-supervised online change detection framework, SSOCD. To the best of our knowledge, this is the first attempt to bridge the unsupervised offline segmentation and supervised online change detection methods, such that changes can be detected online and intendedly. Specifically, as shown in Fig. 5-1, in the SSOCD framework the change detection process is divided into two steps: offline stage and online stage. In the offline stage, unsupervised segmentation method is employed to label change points in unlabeled training data sequence, then these labeled change points are used as training set to train the supervised change detection model. With the learned model, in the online stage, data is processed sequentially and those intended changes concealed in the data stream which are usually captured by offline methods are therefore be detected in an online way.

Algorithm 8 describes the pseudo code of this framework. The first step in SSOCD is to label the desired change points in the training data using designed unsupervised offline segmentation method. Benefit from the global view of the data set, it is achievable to design segmentation method which gives an exact split of the data with changes meet intended requirements, e.g., only detect change in mean greater than a given threshold. In the second step, a significance test is exerted on the derived segments, such that change points labeled by the offline methods are those real changes meet the specified requirements. Depend on the property of intended changes, various statistic significance tests (see Section 5.2.4) as well as customized test can be used to check the quality of the learned results. In the next step, the refined labels are used to train the supervised change detection model. Many online change detection methods can be adapted to the supervised case, such as the supervised

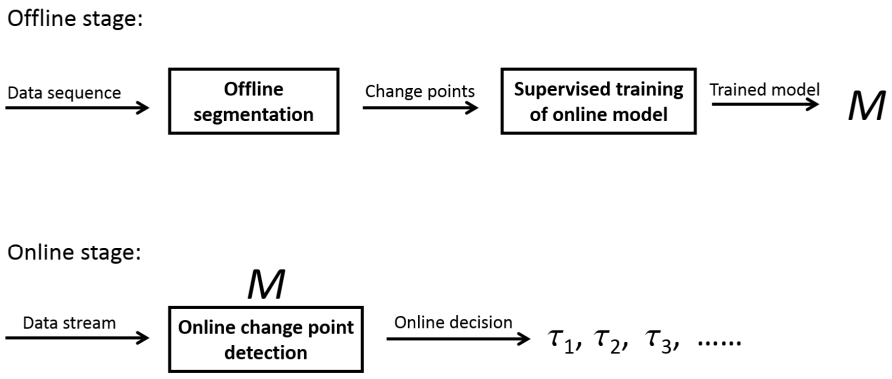


Figure 5-1: Supervised change point detection framework

version of Bayesian online change point detection [162] and supervised CUSUM which uses the criterion given in [64] to maximize the detection power based on labeled change points. In the online stage, the trained model is used to detect changes in the sequential data. Decision about whether a pre-specified change is presented in the data is made each time a new data arrives.

The success of the SSOCD framework depends on the quality of labeled results given by the unsupervised offline segmentation method, and also the power of the supervised online change detection method. In the following, we test several alternatives for both cases, and evaluating their performance on a real-world dataset.

Algorithm 8: Semi-Supervised Online Change Detection, SSOCD

Input: X_{train}, train set;
X_{test}, sequentially arriving test data;
ρ, significance threshold.
Output: M_{online}, online change detection model learned from offline methods
τ, detected change points

Offline Stage:
$\tau_{offline} \leftarrow$ stand_offline_segmentation(X_{train}) //segment the training data offline
$\tau'_{offline} \leftarrow$ significance_test(X_{train}, $\tau_{offline}$) //test the significance of offline result
$M_{online} \leftarrow$ supervised_train(X_{train}, $\tau'_{offline}$, ρ) //train the online CPD model
Online Stage:
$\tau \leftarrow$ stand_online_CPD(M_{online}, X_{test}) //detect change point online
return τ

5.3.3 Offline segmentation methods

Offline labeling of multiple change points can be seen as a process of splitting a non-stationary observation sequence into several contiguous stationary segments. In this section we introduce three typical examples of offline multiple change points estimation as the candidate algorithms for the SSOCD framework, i.e., piece wise constant segmentation based on total variation denoising (TVD), pruned exact linear time (PELT) method, and hierarchical multiple change point estimation (ECP).

Total variation denoising, TVD

In piecewise constant segmentation (PWC) the basic task is to recover a N sample signal m_i from the observed signal x_i, where x_i is corrupted by an additive noise random process e_i, i.e.,

$$x = m + e. \tag{5.10}$$

As stated in [105], most PWC denoising methods can be expressed as the following form:

$$H[m] = \sum_{i=1}^{N} \sum_{j=1}^{N} \Lambda(x_i - m_j, m_i - m_j, x_i - x_j, i - j), \tag{5.11}$$

where x and m are the input output signal of length N, respectively. $x_i - m_j$, $m_i - m_j$, $x_i - x_j$ and $i - j$ are the value differences (d) of input-output, output-output, input-input and sequence index, respectively. Kernel functions (non-negative symmetric functions) and loss functions (non-negative functions) can further be developed on the value differences term d. More detailed explanations can be found in table 1 in [105]. Equation 5.11 assembles the error incurred by every difference into the loss function Λ. $H[m]$ is the summation of Λ over all pairs of indices in the input and output signals to be minimized with respect to the output m.

Among various generalized functions for PWC noise removal, total variation denoising (TVD) has been an intensively explored topic since its first proposal in [142]. The loss function Λ in TVD is:

$$\Lambda = \frac{1}{2}|x_i - m_j|^2 I(i - j = 0) + \gamma |m_i - m_j| I(i - j = 1), \tag{5.12}$$

where $I(S)$ is an indicator function such that $I(S) = 1$ if the condition S is true, and $I(S) = 0$ otherwise. $I(i-j=0)$ selects only terms that have the same index and $I(i-j=1)$ selects only sequentially adjacent terms. γ is a regularization term. As is shown in [105] Eqn. 5.12 is convex and can be solved by methods like quadratic programming, Finite differencing, Coordinate descent, Least-angle regression path follower [157] and Piecewise linear regularization path follower [141, 73]. Time complexity of the TVD depends on the chosen optimization method, normally a complexity of $O(knlog(n))$ (k is the number of maximum number of change points) can be achieved using the LARS/LASSO algorithm.

Pruned exact linear time method, PELT

The PELT is based on the optimal partitioning (OP) method proposed in [170] and [81]. The OP is a search algorithm aims at minimizing

$$\sum_{i=1}^{m+1}[C(x_{(\tau_{i-1}+1):\tau_i}) + \beta], \tag{5.13}$$

where $C(\cdot)$ is the cost function of a segment, m is the number of segments and β is a penalty to avoid over fitting. According to [81] the OP finds the change points in a recursive way. Formally, let $F(s)$ be the minimization of Eqn. 5.13 for data $x_{1:s}$ and $\boldsymbol{\tau}_s = \{\tau : 0 = \tau_0 < \tau_1 < ... < \tau_m < \tau_{m+1} = s\}$ be the set of possible vectors of change points for the data. Set $F(0) = -\beta$. Then $F(s)$ can be written recursively as:

$$\begin{aligned} F(s) &= \min_{\boldsymbol{\tau}\in\boldsymbol{\tau_t}}\{\sum_{i=1}^{m+1}[C(x_{(\tau_{i-1}+1):\tau_i}) + \beta]\} \\ &= \min_t\{\min_{\boldsymbol{\tau}\in\tau_t}\sum_{i=1}^{m}[C(x_(\tau_{i-1}+1):\tau_i) + \beta] + C(x_{(t+1):n}) + \beta\}, \\ &= \min_t\{F(t) + C(y_{(t+1):n}) + \beta\}. \end{aligned} \tag{5.14}$$

Equation 5.14 finds the minimal cost for data $x_{1:s}$ in terms of the minimal cost for data $x_{1:t}$ for $t < s$. For a data sequence of length n, the total time cost for computing $F(s)$ is quadratic in n.

Though the OP provides a more efficient method than SN method, it is still computational costly with a computing time complexity of $O(n^2)$. In the aim of improving the computational efficiency of the OP, [90] proposed a pruning method named PELT with a

time complexity of $O(n)$. This method (PELT) is guaranteed to find the same optimal as the OP. The basic idea for improving efficiency in PELT is to discard those values of change points which can never be the optimal during the optimization process of the OP.

Hierarchical nonparametric multiple change point analysis, ECP

Based on the divergence measure proposed by Szekely and Rizzo [155, 139], James et. proposed a hierarchical nonparametric multiple change point analysis method in [82]. ECP is capable of detecting any type of distributional change in the data, e.g., change in mean, variance, tail etc.. The only assumption that the method exerts on data is, for some $\alpha \in (0, 2]$, the absolute αth moment exists, and data are independent over time.

Suppose $X, Y \in R^d$, and $X \sim F_1$ and $Y \sim F_2$, with characteristic functions $\phi_x(t)$ and $\phi_y(t)$, respectively. The divergence measure which used for determining whether two independent random vectors X and Y are identically distributed is defined as follows:

$$D(X, Y; \alpha) = \int_{R^d} |\phi_x(t) - \phi_y(t)|^2 \omega(t; \alpha) dt \tag{5.15}$$

$$= \int_{R^d} |\phi_x(t) - \phi_y(t)|^2 \left(\frac{2\pi^{d/2}\Gamma(1 - \alpha/2)}{\alpha 2^\alpha \Gamma[(d + \alpha)/2]} |t|^{d+\alpha}\right)^{-1} dt, \tag{5.16}$$

where $\omega(t; \alpha)$ is a positive weight function defined by Matteson and James in [117]. Based on the divergence measure given in Eqn. 5.16, location of multiple change points can be estimated iteratively, either divisively or agglomeratively. Since the divisive way of change estimation shows strong consistency (not the agglomerative way), we only introduce the divisive multiple changes estimation in this thesis. A complete introduction of the agglomerative algorithm and the whole method can be found in [82].

Algorithm 9 presents the procedure of hierarchical divisive change point estimation. At the beginning, a distance matrix based the divergence measure in Eqn. 5.16 is computed for each observation in the data sequence X. Then at each iteration, the most likely change point location is estimated and its statistic significance is tested by a permutation test. If the estimated change point is statistic significant, then the segment it belongs to is divided into two subsegments at the estimated location. Further estimation and division is progressed until no significant change point is found. The process of divisive change point estimation can be viewed as a binary tree. In this tree, the root node represents the situation of no change point, and contains the whole data series. Its children nodes are either the same as

their parent nodes, or correspond to one of the new segments created by their parent nodes by introducing a change point. The time complexity of the ECP is $O(kn^2)$, where k is the number estimated change points [82].

Algorithm 9: ECP, Hierarchical divisive multiple change point estimation [82]

Input: X, data series
p_0, significance level
m, minimum segment length
R, maximum number of permutations for the permutation test
eps, uniform resampling error bound
h, epsilon spending rate h
α, the αth moment with $\alpha \in (0, 2]$.
Output: A segmentation of the data series
Initialize: Initialize the distance matrix $X_{ij}^{\alpha} = |X_i - X_j|^{\alpha}$

```
while Exist a statistically significant change point do
    Estimate the next most likely change point location
    Test the significance of estimated change point
    if Estimated change point is statistically significant then
        Update segmentation
return Final segmentation
```

5.3.4 Supervised Bayesian online change point detection

In this section, we first introduce the basic unsupervised BOCPD model and then show the augmented supervised BOCPD with noisy training labels. Notations in this section are similar to those in [162].

BOCPD was first introduced in [4] and [49]. The basic model is: given a data observation sequence $x_1, ..., x_T$, predict the *run length* at each time step via an *underlying predictive model* (UPM) and a *hazard function* $H(\cdot)$. The run length (r) is defined as the time of no change detected since the last change point. The UPM can be seen as a base model whose parameters η change at each new observation with respect to fixed model hyper-parameters θ_m. The *hazard function* $H(r|\theta_h) \in N \to [0, 1]$ can be thought as the prior probability of a change point occurring given a run length r_t.

Formally, let $x_1, ..., x_t$ denote the observation sequence, r_t be the run length at time t. r_t is incremented at each time step until there is a change point detected after when it is reset to 0. For simplicity, let $r_0 = 0$, meaning there is a change point at $t = 0$, τ_i be the time of the ith change point, therefore, $\tau_0 = 0$. For the hyper-parameters, define

$\theta := \{\theta_m, \theta_h\}$, where θ_m and θ_h are the hyper-parameters of UPM and hazard function, respectively. Further, let $c_t \in \{0, 1\}$ be the change point vector at time t, where $c_t = 1$ if there is a change point at t and 0 otherwise. The relationship between these variables is illustrated in Table 5.2, and their formal relationship can be described as following:

$$r_t = \begin{cases} 0 & , c_t = 1 \\ r_{t-1} + 1, & c_t = 0 \end{cases}, \quad \sum_{t=1}^{\tau_i} c_t = i, \quad c_{\tau_i} = 1. \tag{5.17}$$

Table 5.2: Illustration of relationship between c, r and τ. Each segment is represented using different symbols (▲, ♦, ▼, ★).

t:	0	1	2	3	4	5	6	7	8	9
c:	1	0	0	0	0	1	1	0	0	0
r:	0	1	2	3	4	0	0	1	2	3
τ:	τ_0					τ_1	τ_2			
		▲	▲	▲	▲	♦	▼	★	★	★

Moreover, variables c, r and τ are used to denote latent (true) change points and $\widetilde{c}$, $\widetilde{r}$ and $\widetilde{\tau}$ for the labeled (observed) change points. Given these notations the generative model can be summarized as:

$$\log p(x_{1:T}, c_{1:T}|\theta) = \log p(x_{1:T}|c_{1:T}, \theta_m) + logP(c_{1;T}|\theta_h) \tag{5.18}$$

$$= \sum_{i=0}^{M} \log p(x_{\tau_i+1:\tau_{i+1}}|\theta_m) \tag{5.19}$$

$$+ \sum_{t=1}^{T} \log(H(r_{t-1}+1))c_t + \log(1 - H(r_{t-1}+1))(1 - c_t).$$

Equation 5.19 can be seen as the sum of the log marginal likelihood of x within each segment and the sum of the log probabilities of the run length exhibited by the hazard function $H(\cdot)$.

Conditional on a given run length r_t, the marginal predictive distribution $p(x_{t+1}|x_{1:t})$ can be computed by integrating out the run length variable:

$$p(x_{t+1}|x_{1:t}) = \sum_{r_t} p(x_{t+1}|x_{1:t}, r_t)P(r_t|x_{1:t})$$

$$= \sum_{r_t} p(x_{t+1}|x_{(r)})P(r_t|x_{1:t}), \tag{5.20}$$

where $x_{(r)}$ refers to the last r observations of x, and $p(x_{t+1}|x_{(r)})$ is computed via the UPM. The run length posterior can be computed by normalizing the joint likelihood as:

$$P(r_t|x_{1:t}) = \frac{p(r_t, x_{1:t})}{\sum_{r_t} p(r_t, x_{1:t})}, \tag{5.21}$$

where the joint likelihood $p(r_t, x_{1:t})$ can further be computed online in a recursive way:

$$\begin{aligned}
\chi_t := p(r_t, x_{1:t}) &= \sum_{r_{t-1}} p(r_t, r_{t-1}, x_{1:t}) \\
&= \sum_{r_{t-1}} p(r_t, y_t|r_{t-1}, x_{1:t-1}) p(r_{t-1}, x_{1:t-1}) \\
&= \sum_{r_{t-1}} P(r_t|r_{t-1}) p(x_t|r_{t-1}, x_{(r)}) p(r_{t-1}, x_{1:t-1}) \\
&= \sum_{r_{t-1}} P(r_t|r_{t-1}) p(x_t|r_{t-1}, x_{(r)}) \chi_{t-1}.
\end{aligned} \tag{5.22}$$

Equation 5.22 shows the message passing scheme for recursively computing χ_t from χ_{t-1}. $P(r_t|r_{t-1})$ is the probability over r_t given r_{t-1} in the hazard function. $p(x_t|r_{t-1}, x_{(r)})$ is the predictive distribution in the UPM over the last newly observed data, given the data since the last change point.

Besides the basic constant hazard function $H(r|\theta_h) := \theta_h = c$ ($c \in [0, 1]$ indicates the probability of a change point), other alternatives like logistic hazard:

$$H(t) = h\sigma(at + b), \tag{5.23}$$

where hyper-parameters $\theta_h = \{h, a, b\}$ and $\sigma(\cdot)$ is the logistic sigmoid, are also available. For the UPMs, there are also extensive choices such as Gaussian process time series (GPTS), Auto regressive Gaussian process (ARGP), and Dirichlet process. More detailed work about hyper-parameter learning and the underlying predictive models can be found in [162].

Figure 5-2 gives an illustrative example of the BOCPD method on an artificial data. The artificial data includes seven random generated data segments, where data inside a segment follow a normal distribution. The top panel displays the artificial data (blue dots) and detected change points (red crosses). The bottom illustrates the cumulative distribution function (CDF) of the run length, where the darker the area, the greater the probability of

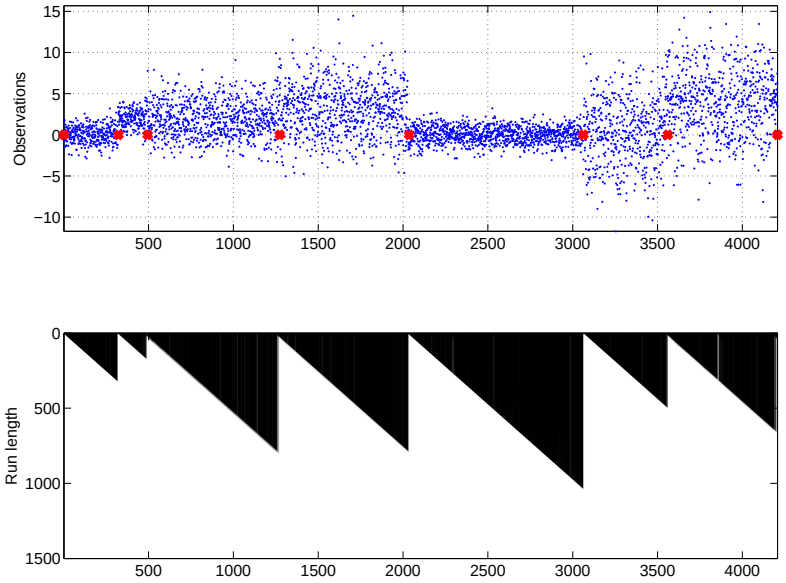


Figure 5-2: An illustrative example of BOCPD on artificial data

corresponding run length[1]. However, as shown in the figure, the problem exhibited by the artificial data is an easy one. As a consequence, the run length distribution area is almost black, implying a strong confidence in the detection result.

Supervised BOCPD The supervised BOCPD was proposed in [162], whose basic idea is to first train a change point model on a data set with labeled change points, and then use the learned model for online detection. More exactly, in the offline step the hyper-parameters of a Bayesian change detection model is learned on a labeled dataset, i.e., given a sequence of training observations $x_{1:T}$ and a corresponding change points label sequence $\widetilde{c}_{1:T}$, learn a model M. Then at the online detection step, new observations are arriving in a stream and the model M is fed with data sequence till the current time t, i.e., $x_{1:t}$, the run length distribution $P(r_t|x_{1:t})$ is computed each time a new observation arrives. One thing to mention is that in the offline training step, noise in the location of the change points might be considered, since the change point labels might be missing, incorrect and spurious.

Generally speaking, there are two ways to train a supervised BOCPD, either generatively or discriminatively. In the former case the hyper parameters θ are learned using generative

[1]At each time index, the probabilities of different run lengths are cumulated in decreasing order of run length, thus an increasing probability (darker color) is observed with the decrease of run length at each time index.

likelihood $p(x_{1:T}, c_{1:T}|\theta)$, and in the latter case the hyper parameters are learned through the discriminative likelihood $P(c_{1:T}|x_{1:T}, \theta)$. Both methods can be augmented with a noise model which considers *jitter* (temporal segmentation error) in the labels. Parameters of the noise model are denoted as θ_n, thus the whole hyper-parameters can be summarized as: $\theta := \{\theta_h, \theta_m, \theta_n\}$.

In the noise model the data sequence $x_{1:T}$ is assumed to be conditionally independent of the label noise. For each change point ϵ_i is used to denote the difference between the observed time of the ith change point $\widetilde{\tau}_i$ (observed) and the ith latent (true) change point τ_i. The zero mean discrete Laplacian (DL) [80] can be used to model the *jitter* $\epsilon_i := \widetilde{\tau}_i - \tau_i \in Z$ as:

$$P(\epsilon_i) = DL(\epsilon_i|\gamma) := \frac{1-\gamma}{1+\gamma}\gamma^{|\epsilon_i|}. \tag{5.24}$$

ϵ_i is assumed to be symmetric with mean $\mu = 0$ and the noise parameters are only the DL dispersion parameter: $\theta_n := \gamma \in [0, 1)$. The "probability of everything" (joint distribution of corresponding variables) in the noisy model is:

$$p(x_{1:T}, \widetilde{\tau}, \tau|\theta) = p(x_{1:T}|\tau, \theta_m)P(\tau|\theta_h)P(\widetilde{\tau}|\tau, \theta_n) \tag{5.25}$$

$$= p_{Gen}\prod_{i=1}^{M} DL(\widetilde{\tau}_i - \tau_i|\gamma), \tag{5.26}$$

where $p_{Gen} = p(x_{1:T}|\tau, \theta_m)P(\tau|\theta_h)$ is used to represent the generative likelihood. Given the noisy model the hyper-parameters θ can be learned by maximizing the $p(x_{1:T}, \widetilde{\tau}|\theta)$ with the latent change points τ integrated out from Eqn. 5.25. More details of how to train a noisy supervised BOCPD can be found in [162].

5.4 Experimental setting

5.4.1 Dataset

As we have shown in Fig. 4-3(a), the failure rate of Biomed dataset exhibits a relatively consistent evolving behavior along the time scale, except only one significant change is present at the 101th time window. However, in order to learn a supervised change detection model, there should be enough normal data as well as the changes in the training set. Therefore we

consider another end-to-end performance monitoring dataset, the IPLab dataset, to verify our proposed work. The IPLab dataset contains performance measurements of large online websites. Specifically, metrics which measure a site's performance (e.g., the page load time on an end-user side) are collected from the end-user point of view. Two types of information are collected for online web sites: *active data* and *passive data*. The *active data* is collected by using performance probes (e.g., probe to test a page's loading time of a site) from probe stations located at the backbone providers every five minutes. On the other hand, the *passive data* is collected from end-user's web browsers by executing specific monitoring script (e.g., a javascript program to collect end-user information on an user's browser) while user visiting a site page.

Compared with the passive data where the diversity of end-user properties might lead to large variance in measuring the site performance, the active data is able to provide a more reliable view of the server's performance during the monitoring, since the probe stations are run on the server side and located at the backbone providers. However, the advantage of passive data lies in its ability to uncover the QoS on the end-user side directly. For this dataset, we use the properties described in Table 5.3 to identify different end-users and target sites. Specifically, we use the tuple $< site, page >$ for the identification of a site, and the tuple $< OS, Browser, DeviceType, Model, Provider, Land >$ for an end-user, i.e., end-users with the same tuple value are classified as the same group.

Table 5.3: Data property description

Property	Comments
Site	Target online site for performance monitoring;
Page	Page of a target site;
OS	Operating System, e.g., Windows, Linux, Android, Apple;
Browser	Browser of the end-user, e.g., IE, Firefox, Chrome, UC Web;
DeviceType	Device type of the end-user, e.g., Tablet, Personal Computer, Mobile, Portable media player;
Model	Model of the device, e.g., 1CPU, 2CPU, 4CPU, ipad, iphone;
Provider	Provider of the network, e.g., Orange, SFR;
Land	Geo-location of the end-user, e.g., Ile-de-France, Rhone-alpes.

The experimental data set was collected between *20-Sep-2013* and *27-Sep-2013*, with a duration of 8 days. In contrast to the active data where samples are collected at a given frequency (every five minutes), the passive data is collected each time an end-user accessing the site. Therefore an obvious characteristic for the passive samples is its high correlation with end-users' daily schedule (e.g., most samples are collected at peak-hours like 9 AM

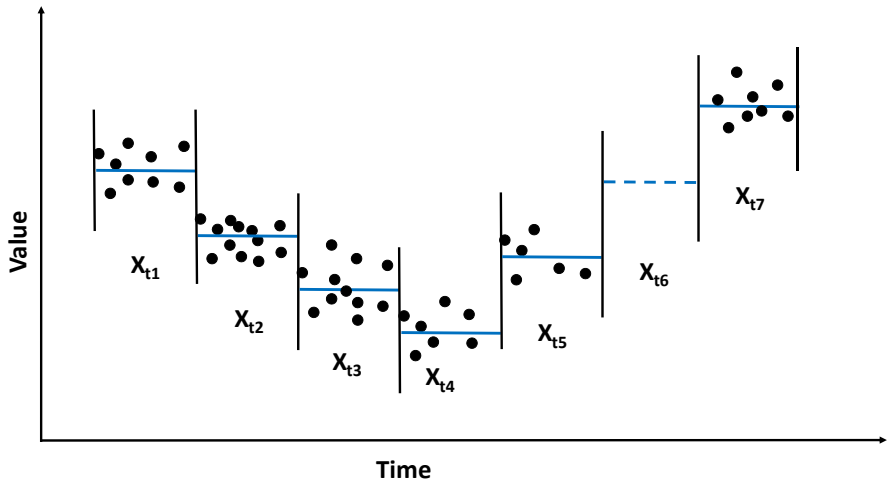
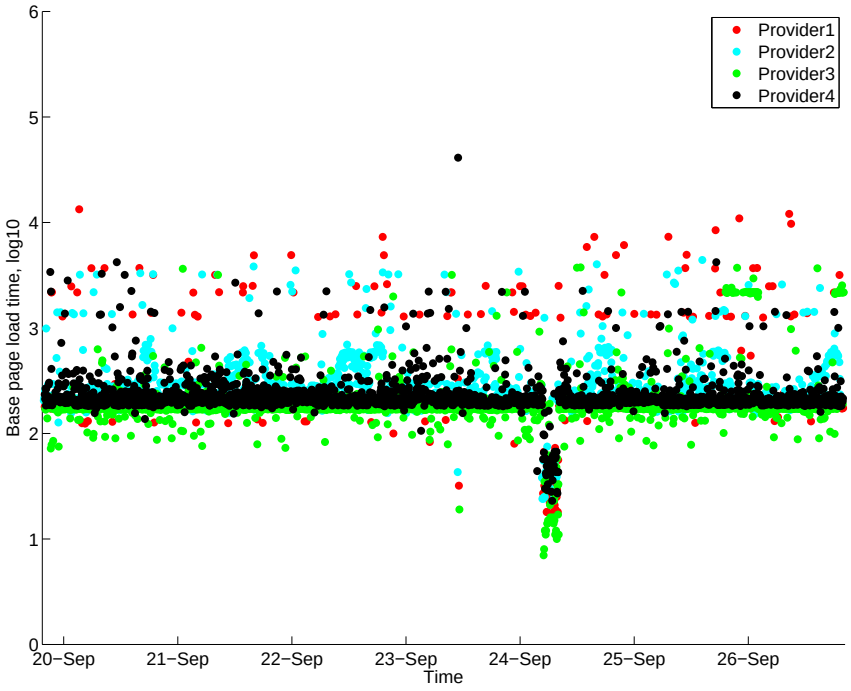


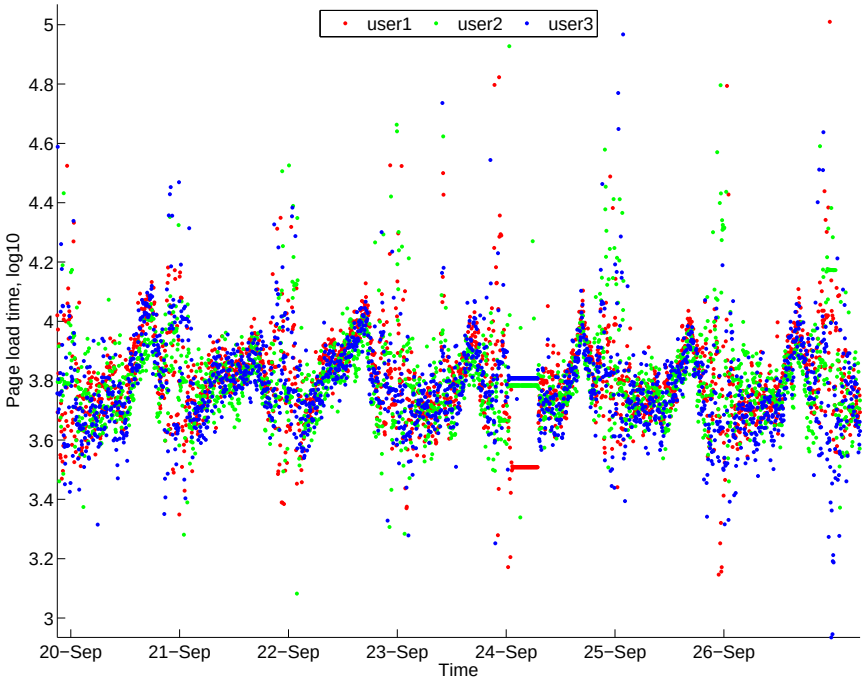
Figure 5-3: An illustrative example of converting raw passive data into time series

and very few samples are gathered at off-peak hours like mid-night). In order to facilitate the comparison, we convert the raw passive data sequence into time series data with a same sample frequency as the active one. Each sample in the new time series is acquired by the median value of all raw data present in the sample interval. Missing values are imputed using the moving average on a length 5 lagging time windows. Figure 5-3 illustrates the transformation process. Within each new time window t, the new sample X_t is taken as the median of all raw values within t (e.g., $X_{t_1}, ..., X_{t_5}$ in the figure). Since no raw observation is presented in time t_6, X_{t_6} is interpolated by the exponentially weighted moving average of its former five values with a damping factor 0.5, i.e., X_{t_6} is derived from $X_{t_1}, ..., X_{t_5}$.

Figure 5-4 gives an example of the active and transformed passive data of a site. The illustrated active data measures the base page load time of a site's page from four probe stations located at different backbone providers, and the passive data describes the full page load time of a site's page of three end-users. As is shown, the active data is approximately piece-wise constant with a small variance and the passive data exhibits a periodic and nonlinear temporal correlations with a large variance.



(a) Active data example



(b) Passive data example

Figure 5-4: Example of *active* and *passive* data

5.4.2 Change detection under the SSOCD framework

Performance evaluation of the SSOCD framework is measured on both the active and passive example data. To conduct a semi-supervised change detection, three offline segmentation methods are evaluated on the dataset, and the one with the best performance is selected to train the supervised online algorithms. Throughout this chapter, we use the ks-test with a threshold 0.1 to evaluate the significance of the offline result (line 3 in Algorithm 8). Refined change points (labels) are used to train the online change detection algorithms. Comparison between the unsupervised and supervised online change detection algorithms is carried out on both datasets.

For the active dataset, we select three data series of a target site, with each series representing the probing result of a probe station located at a different network provider. For the passive dataset, we select three user data series with the largest number of observations in the raw series. Each data series is further split into two parts: the first 1000 observations as training set for parameters learning and the left observations as test set for performance comparison. Each of the data series is standardized, with the mean and scale estimated *only* from the training data. For the performance evaluation, we use the asymptotic p-value of the ks-test to measure the discrepancy between neighboring segmentations. The p-value can be seen as the probability of observing the given samples, or one more extreme, under the null hypothesis that data in two different sets are from the same continuous distribution. Hence a large p-value indicates an inclination to accept the null hypothesis and a small p-value implies a rejection.

Three different offline segmentation methods were compared on the active data: PELT, ECP, TVD. Final parameter settings of each algorithm are listed in Table 5.4[2].

For the supervised online change detection method, we test the BOCPD (Section 5.3.4) with two UPMs: the IFM (Independent factor model with a changing mean function for change detection) and the GPCP (Gaussian process time series designed for change point detection). The IFM is appropriate for modeling the piece wise constant data series, while the Gaussian Process based change point detection method (GPCP) is capable of modeling the nonlinearity within data series intrinsically, help us dealing with periodicity, short term correlations, long term trend and etc. Thus in order to highlight our emphasize on the semi-

[2]Note that for 'PELT' and 'ECP' we use the R packages of *changepoint* [90], *ecp* [82], respectively, and for 'TVD' we use the method in [105].

supervised learning framework, we use the IFM to model the piece wise constant active data and the GPCP to describe the nonlinear passive data series. [162] gives a more detailed introduction about the GPCP and the IFM.

The two BOCPD models use the logistic hazard function (Eqn. 5.23), with corresponding parameters learned during training. A rational quadratic (RQ) covariance function together with a constant covariance is used for the GPCP. The hyper-parameters of GPCP are learned on the training set with five runs, choosing the one with a best function value.

Due to the similarity between different data series (e.g., as delivered in Fig. 5-4, data series of different provider have a similar trend and distribution in the active data), experimental result on the selected data series share a similar pattern. Therefore, for simplicity, we only show the result of a single data sequence for each active and passive dataset in the following section.

Table 5.4: Offline algorithm parameters setting

	parameters

PELT	penalty='Manual'[3];
	penalty.value=60;
	method='PELT';
	test.stat='Normal'.

ECP	significance level=0.05;
	Max number of random permutation=50;
	moment index=1;
	min segment size=40.

TVD	lambda=1;
	max iteration=60;
	stop tolerance=1e-3.

5.5 Experimental results

5.5.1 Active data

Offline segmentation Figure 5-5 shows the segmentation results of three offline algorithms on the active dataset, with vertical lines indicating the detected change point positions and

[3]Because there are too many noisy change points in the results of using an auto penalty of either 'SIC' or 'AIC', here we use the 'Manual' penalty for PELT instead. Penalty value is selected on the first 1000 observations.

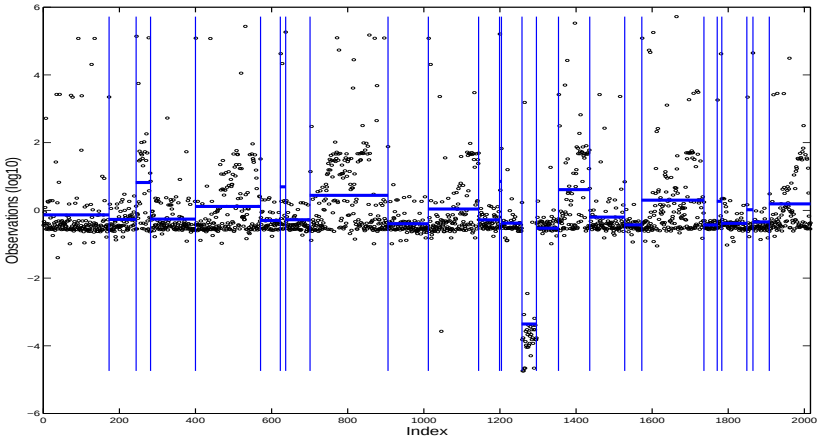
horizontal bar representing the segment means. Corresponding p-values of the ks-test at the significance level of 0.05 for the neighboring segments are given in Fig. 5-6.

The PELT segments the data sequence into 26 subparts, with some of which being very short, e.g., the 7th, 13th, 22nd, and 24th segment have lengths of 14, 5, 12, and 16, respectively. Moreover, the intra-segment difference is not always significant in the result of PELT. For example, among the 25 detected change points, 7 of them have the p-value of a ks-test with 5% significance level for consecutive segments greater than 0.1, indicating the hypothesis that samples in two consecutive segments are from the same continuous distributions can not be rejected at the specified significance level. Similar result can also be found in TVD. The TVD splits the data sequence into 26 segments. The p-values of the ks-test for the segmentation are illustrated in Fig. 5-6(c). Among the 26 detected change points, 7 of them are associated with a p-value greater than 0.1.

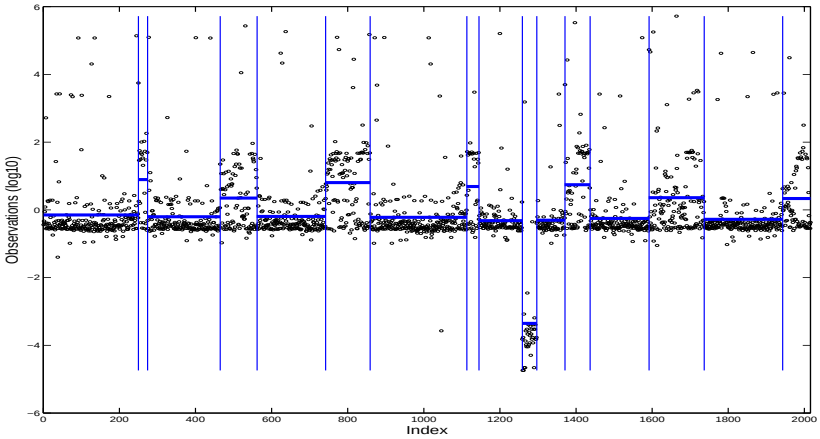
On the contrary, the segment-wise difference of result of ECP is quite remarkable, whose p-values of the ks-test between segments are all less than $1e-6$ (Fig. 5-6(b)). In other words, the hypothesis that samples in consecutive segments are from different distribution can not be rejected. To sum up, on the active data set, the ECP method outperforms PELT and TVD, in terms of intra-segment difference; and is more appropriate for estimating the number of change points and their corresponding locations in practice.

Unsupervised IFM Result of unsupervised BOCPD with IFM on the active dataset is illustrated in Fig. 5-7(a), with corresponding p-values of the ks-test shown in Fig. 5-7(b). Standardized data sequence (blue) is shown in the top panel in each figure, and the proposed change points (green lines and red crosses) are illustrated in the middle panel. The vertical green lines mark the locations of change points proposed by offline segmentation and the red crosses mark the locations of change points detected by online algorithms, with a probability under the BOCPD posterior exceeds a threshold mentioned in the figure's caption. The bottom panel in each sub-figure shows the run length CDF (cumulative distribution function) and its median (solid red).

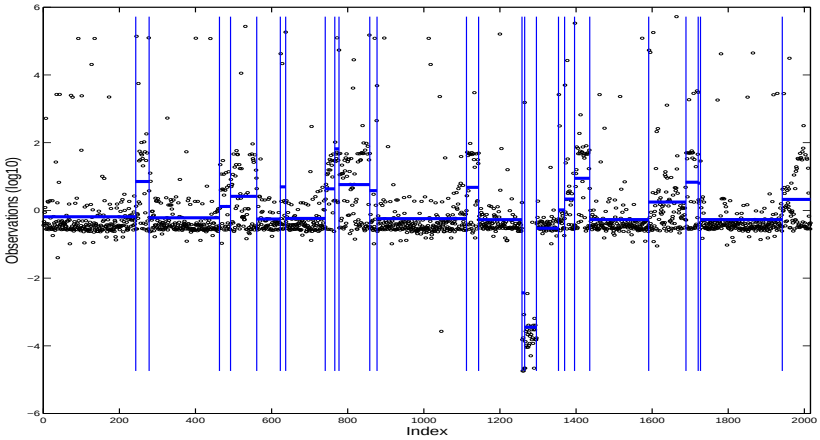
As we can see from the run length distribution, IFM gives very spiky median run length, implying a low prediction confidence in the results given by underlying models. Moreover, the average median run lengths (red curve in each sub-figure's bottom panel) of it is very short, with the maximum median run length of IFM is less than 40 (Fig. 5-7(a)). As a consequence, there are many proposed change points with a probability under the BOCPD



(a) PELT

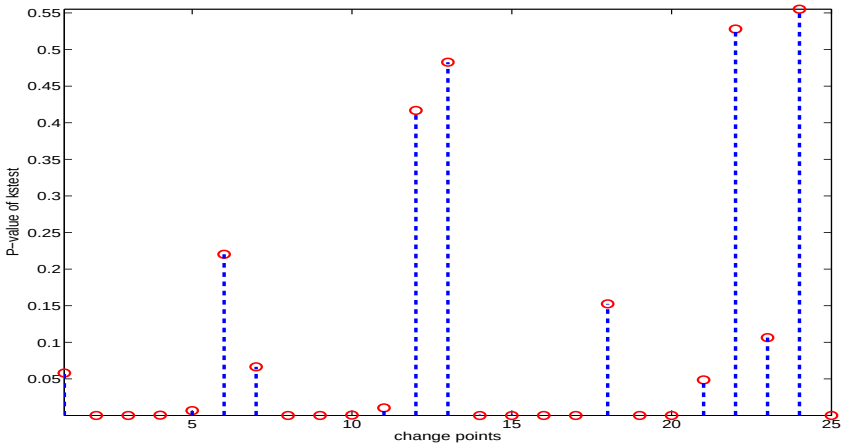


(b) ECP

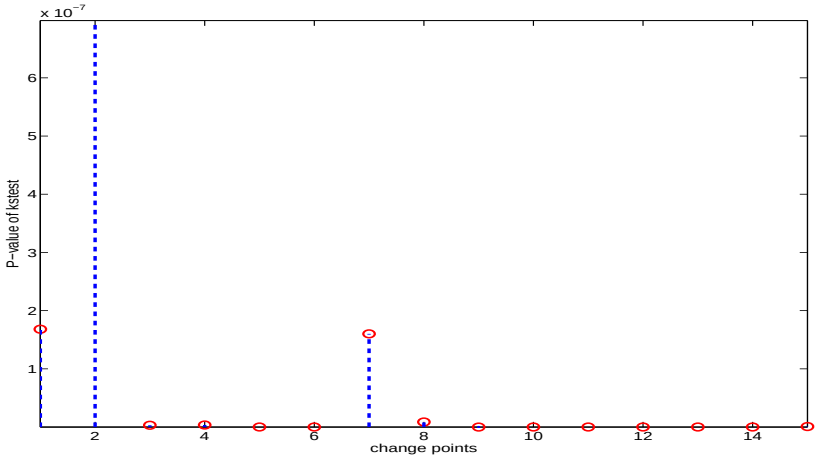


(c) TVD

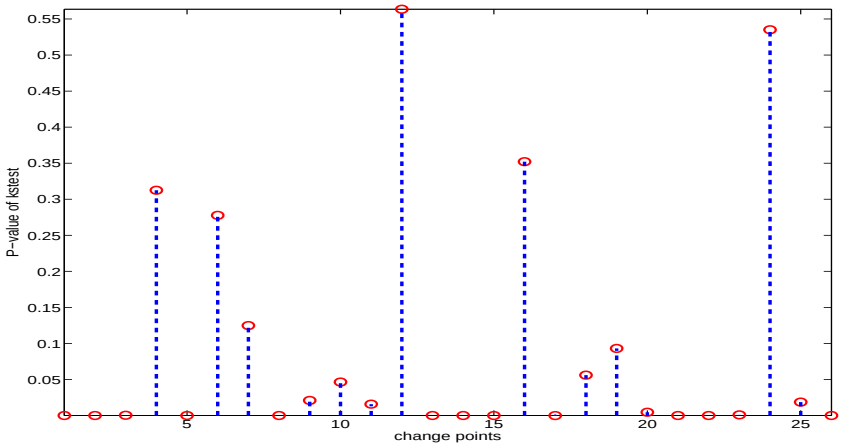
Figure 5-5: Offline segmentation on active data



(a) PELT



(b) ECP



(c) TVD

Figure 5-6: P-values of Kolmogorov-Smirnov test for neighbouring segmentations of offline results, active data

posterior exceeding 0.96, which in other words, compared with the offline methods the unsupervised IFM is very sensitive.

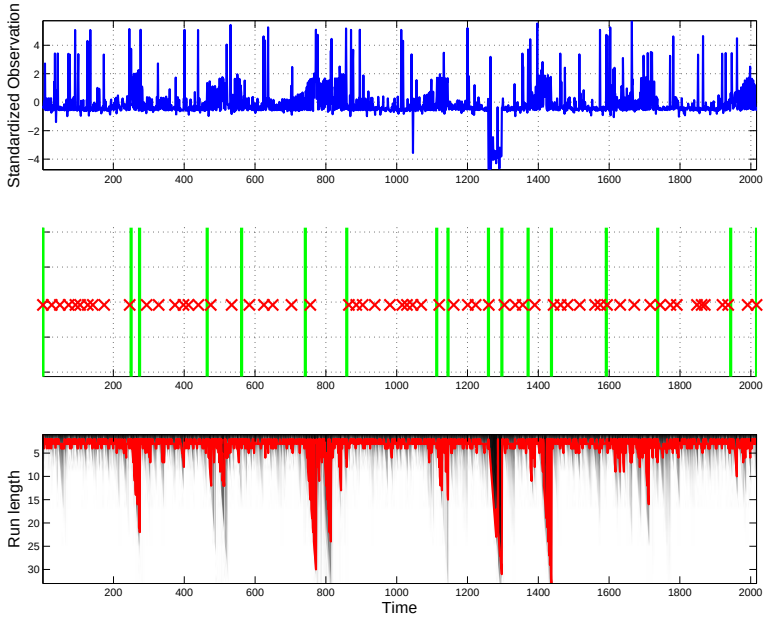
The reason responsible for this phenomena lies in the data sequence. Without those outliers (i.e., the spiky blue line in the raw sequence) the standardized data sequence is a clean piece wise constant with a zero mean and a relatively small variance. Changes proposed by the offline segmentation method clearly exhibit the changes between constant pieces of the clean sequence. These changes represent the macro variation in the data. They are easy to be detected and are actually detected by the unsupervised IFM, since for almost every change given by the offline method (green line) is associated with a candidate change point proposed by the unsupervised method (red cross). However, besides the macro changes those transient ones are also captured by the unsupervised methods. The IFM tends to raise a change point each time there is an outlier (Many spiky vertical blue lines in data sequence are associated with a red cross).

The p-values of ks-test between neighboring segments of the unsupervised IFM are shown in Fig. 5-7(b). On this active dataset, IFM proposes 59 change points, with 21 of which have p-value less than 0.1 and the others are greater than 0.1. Since the p-value is concerned with the probability of observing samples in two segments from a same continuous distribution, those change points with p-value greater than 0.1 can thus be seen as non significant ones.

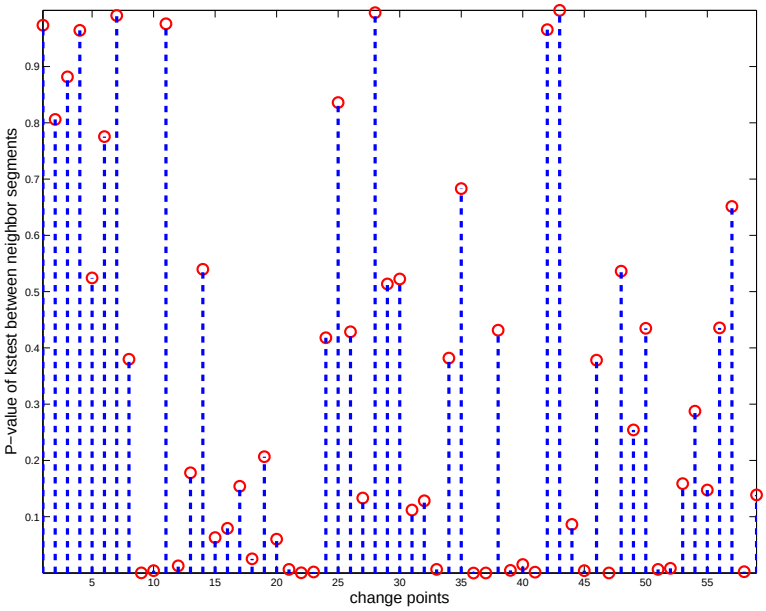
To sum up, the trivial and transient changes proposed by the IFM exposes its sensitivity and vulnerability to noise and outliers. Consequently, a high false alarm rate is anticipated for both methods.

Supervised IFM Performance evaluation of the supervised IFM on the active data is illustrated in Fig. 5-8. Compared with the unsupervised versions, run length distributions of supervised IFM is much more smooth. The average median run lengths of IFM becomes longer, e.g., the maximum median run length of IFM is about 250 (The peaked median run length of supervised IFM around time index 1250 is not a valid one, because it does not grow gradually to the peak value). The smooth and dark run length distributions exhibit the high confidence in the predictions. As a consequence, for the supervised IFM the number of proposed change points with a probability under the BOCPD posterior exceeding 0.96 decreases obviously, from 59 to 14.

The p-value of the ks-test on neighboring segments in Fig. 5-8(b) verifies the improved result. Contrast to the unsupervised case, in supervised IFM only one change point is



(a) Run length distribution



(b) p-value of ks-test

Figure 5-7: Run length distribution of unsupervised BOCPD (IFM) on active dataset, with probability threshold 0.96. In the middle panel of Fig. 5-7(a), red crosses represent change points detected by online algorithm and green lines represent change points proposed by offline method. A same representation is used in the following figures.

Table 5.5: Similarity between results of online and offline methods on active test set

	Adjusted Rand index	Rand index
Unsupervised IFM	0.3317	0.9333
Supervised IFM	0.8006	0.9701

observed to have a p-value greater than 0.1. This attest to the conclusion that most of the change points proposed by supervised IFM are statistically significant. The comparison of median run lengths for unsupervised and supervised IFM is shown in Fig. 5-9, where a striking contrast is exhibited on the two methods' results. In contrast to an oscillating and transient median run length distribution demonstrated by the unsupervised method, the supervised IFM delivers a smooth and long term run length growth. Accordingly, a stronger confidence in predictions of supervised IFM is expectable.

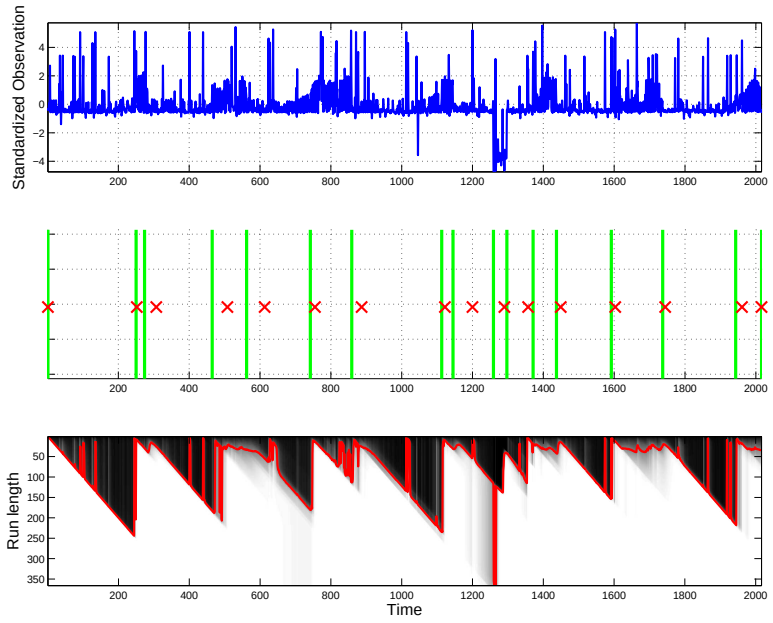
The Rand Index and its adjusted version evaluate clustering similarity by assessing the segment membership of pairs of data points. In this section, we use the results given by ECP as the base model, and compute the rand index (adjusted rand index) by comparing its segmentation result and the unsupervised/supervised IFM. Table 5.5 illustrates the comparison between the two IFMs. In contrast to unsupervised IFM whose adjusted rand index is low, the supervised IFM shows a much higher adjusted rand index, attesting to the effectiveness of supervised learning in the framework.

5.5.2 Passive data

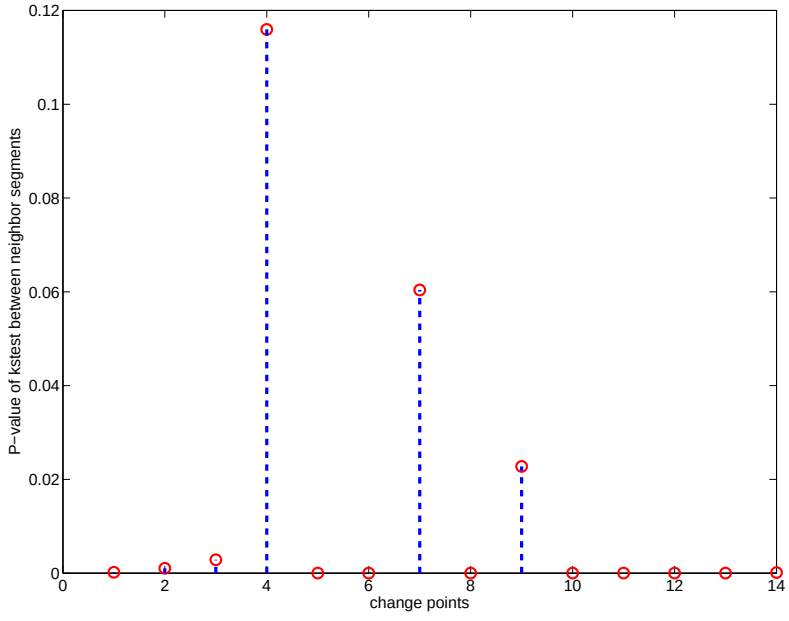
This section gives the experimental results of various methods on the passive data. As we show in previous section, ECP gives much better segment results than the PELT and TVD, hence we only employ ECP to segment the passive data sequence. The parameters setting for ECP is the same as in the active experiment.

Figure 5-10(a) illustrates the partition result of ECP and Fig. 5-10(b) shows the corresponding ks-test result. ECP proposes 27 change points on the passive data, where 26 of them are statistically significant according to their ks-test results and one is treated as insignificant as it has a p-value (0.12) of the ks-test greater than the threshold 0.05.

Unsupervised GPCP Performance evaluation of unsupervised GPCP is shown in Fig. 5-11. The data sequence is split into 32 segments with a posterior probability greater



(a) Run length distribution



(b) p-values of ks-test

Figure 5-8: Result of supervised BOCPD (IFM) on active dataset, with probability threshold 0.96.

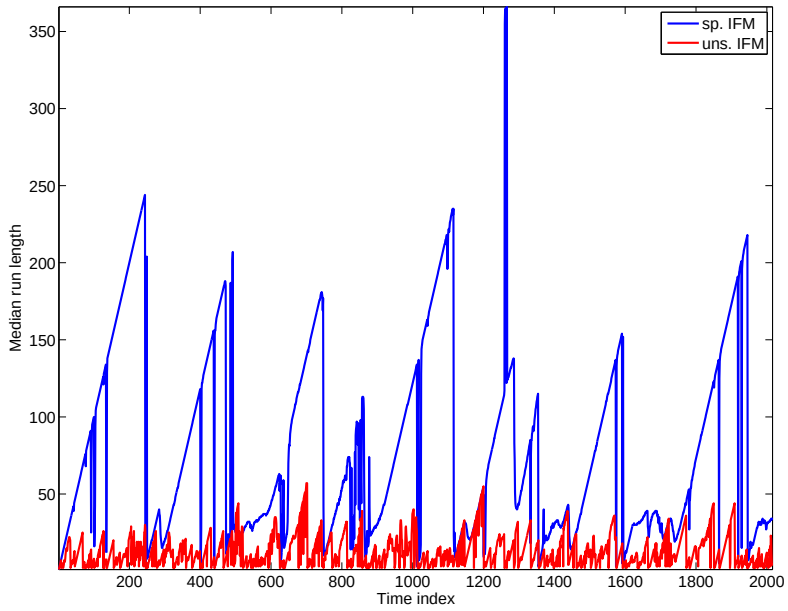
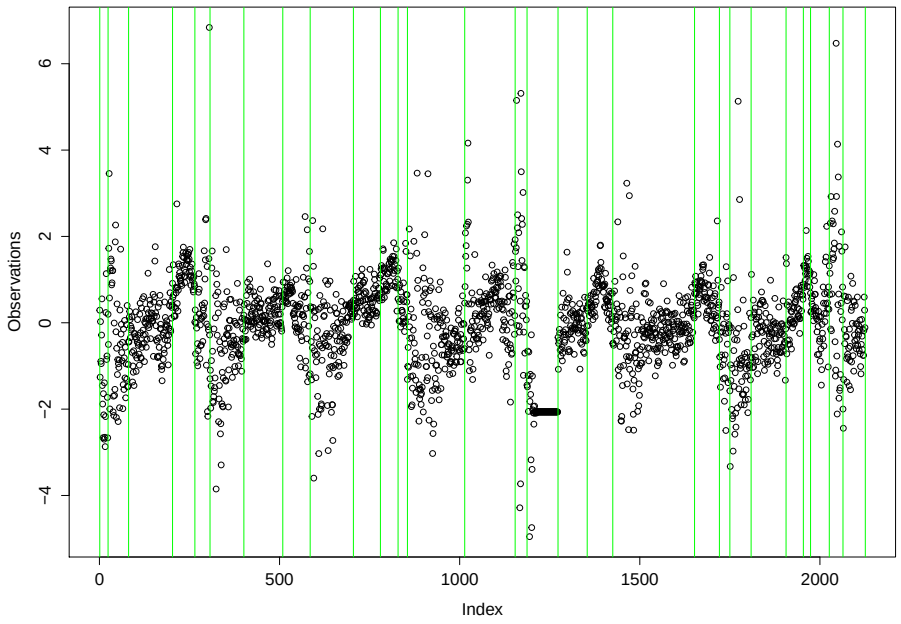


Figure 5-9: Median run length comparison between unsupervised and supervised IFM on active dataset

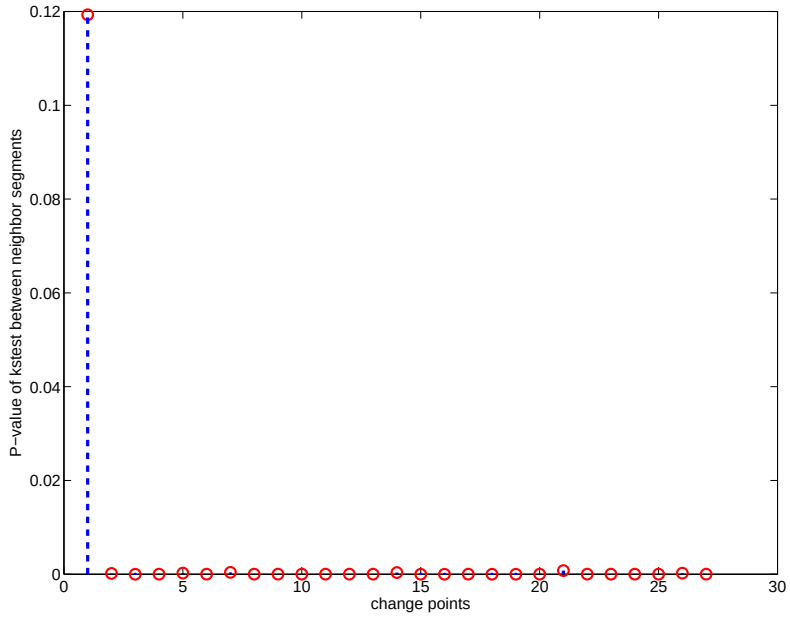
than 0.96 (threshold). In contrast to the unsupervised IFM on the active dataset, the unsupervised GPCP exhibits a stronger confidence in the run length distribution (smooth median run length curve). Most of the proposed change points are relatively significant, as most of the p-values of the ks-test for neighboring segments are less than 0.1 (Fig. 5-11(b)). Nevertheless, there are two change points do not deliver convincing results, i.e., the 15th and the 31st change points have p-values greater than 0.1. In other words, observations from segments separated by these two change points have a relative high probability of generated from the same distribution.

Despite the good prediction performance of unsupervised GPCP, a clear fluctuation is exhibited on its median run length curve (red line in the bottom panel in Fig. 5-11(a)). The jagged shape of this curve implies a fluctuation in the prediction confidence. In a word, the unsupervised GPCP is able to capture the nonlinear correlation and detect those changes in the data sequence. A notable inter-segments dissimilarity is shown in the derived segmentations, but a clear fluctuation is concealed in the run length distributions as well.

Supervised GPCP The hyper-parameters of supervised GPCP are learned on the training set using the segmented result of ECP as labels, and performance evaluation is

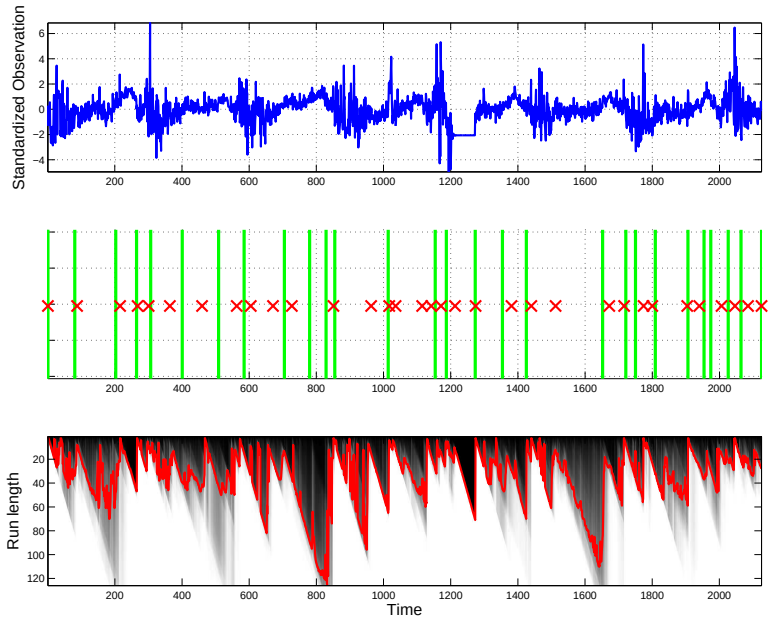


(a) Segmentation result

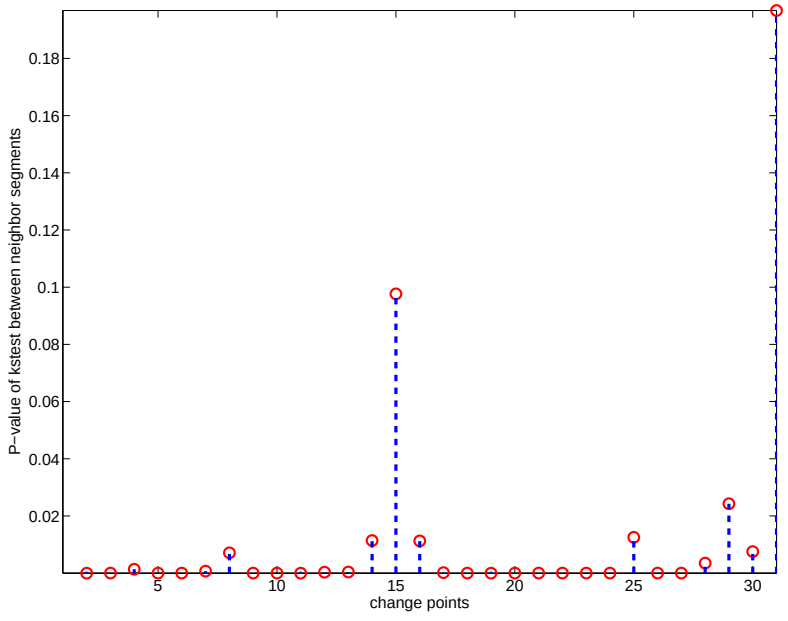


(b) p-values of ks-test

Figure 5-10: ECP on passive data.



(a) Run length distribution



(b) p-value of ks-test

Figure 5-11: Run length distribution of unsupervised BOCPD (GPCP) on passive data, with probability threshold 0.96

Table 5.6: Similarity between results of online and offline methods on passive test set

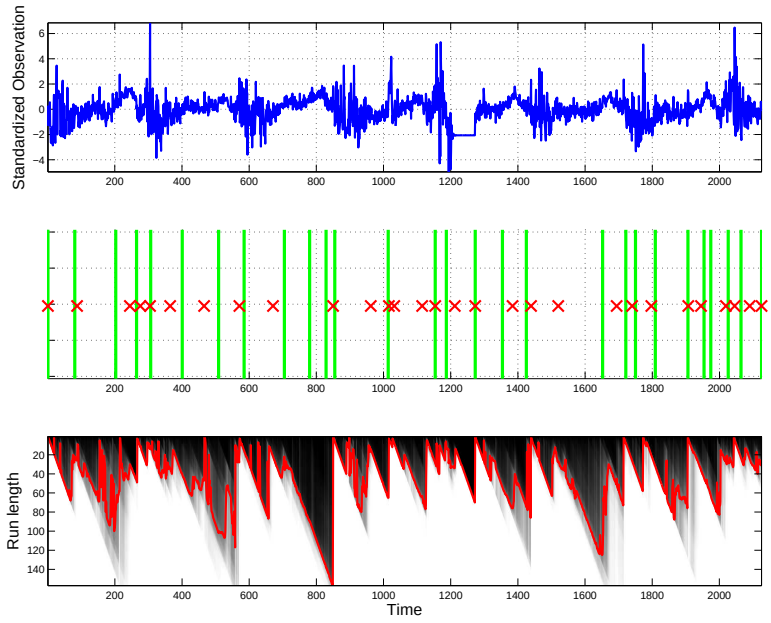
	Adjusted Rand index	Rand index
Unsupervised GPCP	0.6002	0.9405
Supervised GPCP	0.5991	0.9395

carried out on the test set. The first change point proposed by ECP is removed from the training set, since it exceeds the p-value threshold in the significance test. Figure 5-12 illustrates the result of supervised GPCP on the passive dataset. The data sequence is split into 28 partitions under a probability threshold 0.96, less changes than in the unsupervised case. The p-values of ks-test for the segmented result are shown in Fig. 5-12(b), where three of the proposed change points have a p-value close to or greater than 0.1.

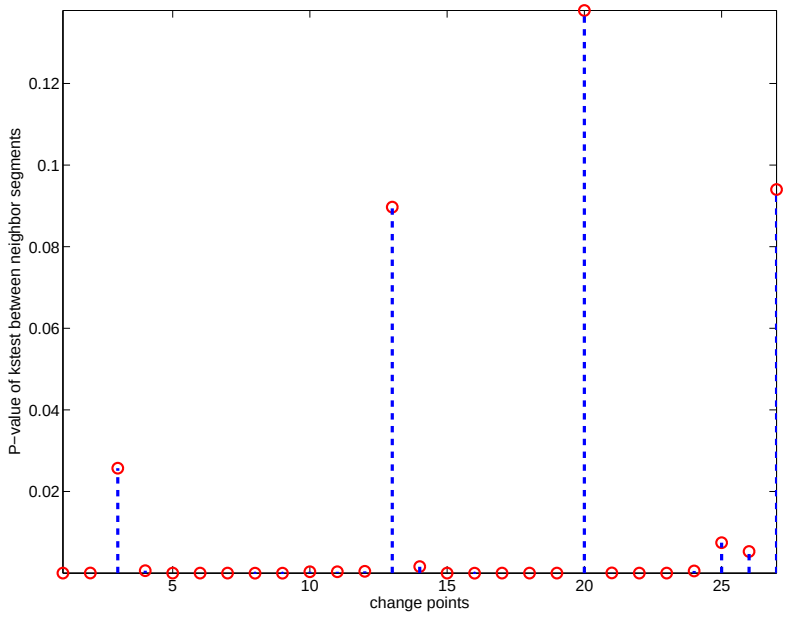
Another difference of the two GPCPs lies in the derived run length distributions. As shown in Fig. 5-13, the median run length curve of the supervised GPCP is always above that of unsupervised GPCP, indicating a stronger in its predictions. Besides, as shown by the four marked green rectangle area in the figure, the result of supervised method is smoother than that of its counterpart. This observation also confirms the improvement in prediction confidence.

To compare the similarity of online methods against the offline alternative, we use the partitions of ECP as the base model to compute the rand index (adjusted rand index) of unsupervised/supervised GPCP. Table 5.6 illustrates the results of two GPCPs. Different to the result on active dataset, difference between the unsupervised and supervised methods is inapparent. This is because the change points proposed the two methods are actually similarly distributed. The adjusted rand index between the two GPCPs is 0.850 on the test set.

In summary, compared with the unsupervised GPCP, the supervised GPCP is capable of sharpening as well as smoothing the run length distribution and median run length, such that the underlying prediction confidence is enhanced. However, the inter-segments' difference of the supervised GPCP's result does not show an advancement.



(a) Run length distribution



(b) p-value of ks-test

Figure 5-12: Run length distribution of supervised BOCPD (GPCP) on passive data, with probability threshold 0.96

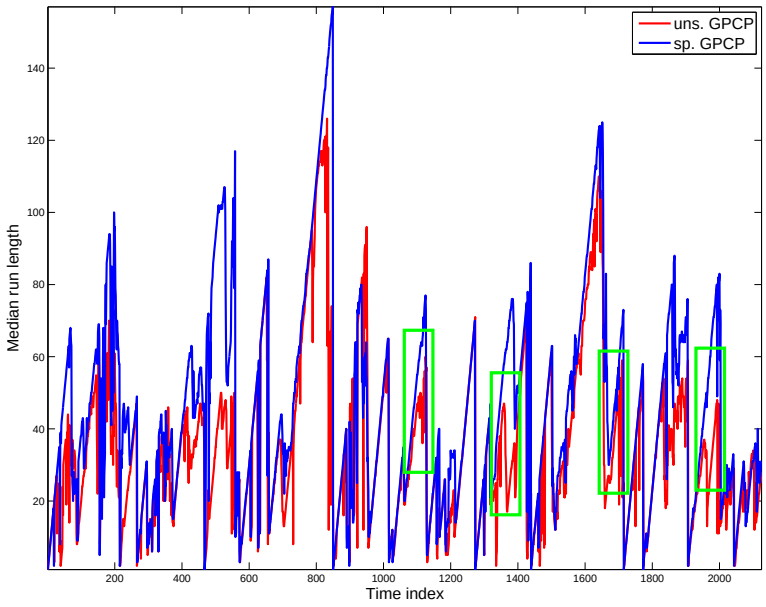


Figure 5-13: Median run length comparison between unsupervised and supervised GPCP on passive dataset. The green rectangles highlight the difference in median run length between the two methods.

5.6 Conclusion

There are different approaches in which change point detection can be performed, from sequential (online) change analysis to retrospective (offline) data segmentation. However, making a decision about changes in a data sequence either online or offline is determined by the practical demand of an application. In this chapter, we are mainly concerned with the online change detection problem. We propose our attempt to bridge the offline and online methods by using the SSOCD framework, within which the unsupervised offline approaches are employed to producing change point labels for training a supervised Bayesian online change detection model, and the trained model is then engaged in detecting changes online.

In such a way, we are aiming at attacking two problems for supervised online change detection: the shortage of labeled change points and the undesired detected changes. The former problem is solved by learning labels using unsupervised offline methods on the raw data set. In the offline case, the whole data set is available for involved method, thus an optimal partition of the data sequence meeting specific requirements is possible if the segmentation procedure is well designed. The later problem can be alleviated by choosing

intended labels for training the supervised model. Intended change points can be obtained either by designing specific offline method or by filtering the result of offline method.

The usefulness of the proposed framework is verified on a real world dataset. Specifically, we compared three state-of-the-art offline segmentation methods, i.e., ECP, PELT, TVD, on the dataset. Because of the significance of its proposed change points, ECP is selected as the offline method to provide labels for the supervised training. The supervised Bayesian online change detection model is trained on the derived labels. Two underlying prediction models (UPM), IFM and GPCP, are employed to model the linear and nonlinear characteristics in two different data, respectively. The result given by the supervised IFM on a piece wise constant data demonstrates the effectiveness of the framework, a remarkable improvement against unsupervised version both in prediction accuracy and confidence. A similar advancement in prediction confidence is also presented in the result of supervised GPCP on a nonlinear dataset. However, the improvement in prediction accuracy is not clearly exhibited, instead the supervised GPCP shows a similar segmentation to its unsupervised counterpart.

Chapter 6

Conclusions and Perspectives

Reliability of large scale production grids and clouds is always the main concern both from the system administration and end-users' experience view. Instead of building a complete a priori knowledge of the software and hardware infrastructures as in most conventional detection or diagnosis methods, we propose to use appropriate techniques to perform end-to-end fault monitoring for such large scale systems, leaving the inaccessible details of involved components in a black box. The way we progress our research can be summarized by a series of assumptions about the temporal behavior of the data: 1) *static*, 2) *stationary*, 3)*non-stationary*, with each assumption corresponds to work in Chapter 3, 4 and 5, respectively. In the following, we first summarize the key issues touched in previous chapters in the order of employed assumptions, and then discuss the main research perspectives opened for future work.

6.1 Summary

- **Static** Chapter 3 considers the temporal behavior of data as an irrelevant factor in a given period, and performs the prediction task on a collapsed observation matrix. Specifically, the collaborative prediction is employed as a scalable and promising strategy to extract hidden information from the monitoring data with a limited intrusiveness to the target system. In contrast to recommendation context, the monitoring task embraces a decisive advantage on the ability of adaptively building intended knowledge. Effectiveness of a combination of Collaborative Prediction and Active Learning has been demonstrated on a large dataset collected from a production grid (EGI).

Basically, two key issues in the fault prediction problem are thoroughly explored with an active strategy based on min-margin prediction heuristic: the imbalance of positive and negative examples (in real systems faults are always the minority group), and the transient faults. The internal prediction model of Active Probing is updated with the adaptive acquisitions of new knowledge, such that hidden information of monitoring data is uncovered iteratively and progressively.

- **Stationary** Chapter 4 assumes an implicit stationary model on the temporal behavior of the data, i.e. information from the past can benefit the prediction for the future. Specifically, the sequential correlations between consecutive data observations are explored using the sequential matrix factorization (SMF) method, which exploits both the spatial knowledge (information from collaborators) and temporal knowledge (information from the past) concealed in the data. Moreover, Active learning is also employed in combination with the SMF (SMFA) in the aim of alleviating the imbalance data class problem and the transient fault problem. Besides, through practice, a smoothing action taken on the estimation sequence of a method has shown to be beneficial to improve the prediction performance. Through the comparison of SMF/SMFA with several basic methods like EWMA, SSVD, MMMF and TENSOR on a large sequential dataset, we have exhibited the strength of SMF/SMFA. A practical conclusion of experiments on the stationary assumed sequential dataset is SMFA with a smoothing action provides a promising solution for the sequential end-to-end fault prediction task, however, in the presence of abrupt changes an utter degradation of prediction performance is unavoidable. (See the 101th time window example in Section 4.4.2). This directs us to the research of the next topic: detecting changes in the observation sequence.

- **Non-stationary** In the aim of detecting abrupt changes in the observation sequence of end-to-end fault monitoring, Chapter 5 assumes a non-stationary model on the temporal behavior of the data. A semi-supervised online change detection framework, i.e. the SSOCD, is proposed to bridge the unsupervised offline and supervised online approaches. Specifically, in SSOCD the unsupervised offline segmentation approaches are employed to producing change point labels to train a supervised Bayesian online change detection model, and the trained model is then engaged in detecting changes online. In this way, two problems in supervised online change detection are under

target: the shortage of labeled change points and the undesired detected changes. The former problem is solved by learning labels using unsupervised offline methods on the raw data set. With a global view of the whole data set available for offline method, an optimal partition of the data sequence meeting specific requirements becomes possible if the segmentation procedure is well designed. The later problem is alleviated by selecting intended labels for training the supervised model. Intended change points can be obtained either by designing specific offline method or by filtering the result of offline method.

The effectiveness of the proposed framework is verified on a real world performance monitoring dataset. Specifically, three state-of-the-art offline segmentation methods, i.e. ECP, PELT, TVD, as well as the supervised Bayesian online change detection model with two underlying prediction models (UPM), i.e. IFM and GPCP, are tested in this framework. The result given by the supervised IFM on a piece wise constant data demonstrates a remarkable improvement against the unsupervised version both in prediction accuracy and confidence. A similar advancement in prediction confidence is also presented in the result of supervised GPCP on a nonlinear dataset. However, the improvement in prediction accuracy is not clearly observed, but the supervised GPCP shows a similar segmentation to its unsupervised counterpart.

6.2 Perspectives

This thesis provides many opportunities for future work, both algorithmic and applicative. To begin with, as we have shown in Section 4.4.2, due to the assumption of stationary data sequence, the presence of abrupt changes in the data will result in an evident degradation of the performance of prediction algorithms. A valuable direction would be to integrate the change detection framework and the sequential fault monitoring method, such that if a significant change is detected in the data sequence, the fault monitoring model can be rebuilt.

Another direction for further research considers the sampling strategy in sequential prediction. Currently, samples in SMFA are selected with two strategies: the most uncertain predictions in the last run guides the selection of samples to enhance the current prediction confidence and the random sampling strategy to avoid an overfitting of the past knowledge.

The current sample ratio between the two strategies is fixed and set to $1:1$. However, given this exploitation (benefits lie those most uncertain predictions) and exploration (random samples) problem, a straightforward extension is to consider this problem under the sequential decision optimization framework. Hybrid optimization indicator mentioned in [164] can be considered to make a balance between the exploitation and exploration trade-off.

Besides the above directions, the unclear theoretic properties like convergency and consistency of the SSOCD framework remains a major concern for further study. Specifically, a few questions of the SSOCD may be proposed: given concrete an offline segmentation method and a supervised online change detection method, what is the theoretic error bound between the two? If the result of an offline method is consistent, under what condition will the supervised online approach has the same property?

A long term applicative direction is to study how the SSOCD framework can be applied in to various online change detection applications: intrusion detection, target detection in video stream and etc. Namely, the task to design a specific offline approach to learn the labels of intended changes or targets, and train a supervised online detection model based on the learned labels. After then the hidden changes or targets can be detected by the trained model in an online manner.

Bibliography

[1] Activemon, https://www.oar.net/initiatives/itecohio/activemon, November 2013.

[2] Nagios, http://www.nagios.org/, November 2013.

[3] Evrim Acar, Daniel M Dunlavy, and Tamara G Kolda. Link prediction on evolving data using matrix and tensor factorizations. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*, pages 262–269. IEEE, 2009.

[4] Ryan Prescott Adams and David JC MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007.

[5] Sandip Agarwala, Yuan Chen, Dejan Milojicic, and Karsten Schwan. Qmon: Qos- and utility-aware monitoring in enterprise systems. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 124–133. IEEE, 2006.

[6] Reena Aggarwal, Carla Inclan, and Ricardo Leal. Volatility in emerging stock markets. *Journal of Financial and Quantitative Analysis*, 34(1), 1999.

[7] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research (JMLR)*, 9:1981–2014, 2008.

[8] Hirotugu Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.

[9] EC Amazon. Amazon elastic compute cloud (amazon ec2). *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[10] Elena Andreou and Eric Ghysels. Detecting multiple breaks in financial market volatility dynamics. *Journal of Applied Econometrics*, 17(5):579–600, 2002.

[11] Sergio Andreozzi, Stephen Burke, Felix Ehm, Laurence Field, Gerson Galang, Balazs Konya, Maarten Litmaath, Paul Millar, and JP Navarro. Glue specification v. 2.0. In *Open Grid Forum Recommendation Documents*. Open Grid Forum, 2009.

[12] Sergio Andreozzi, Augusto Ciuffoletti, Antonia Ghiselli, Demetres Antoniades, Michalis Polychronakis, Evangelos P Markatos, and Panos Trimintzios. On the integration of passive and active network monitoring in grid systems. In *Integrated Research in GRID Computing*, pages 147–161. Springer, 2007.

[13] Yupaporn Areepong. Explicit formulas of average run length for a moving average control chart for monitoring the number of defective products. *International Journal of Pure and Applied Mathematics*, 80(3):331–343, 2012.

[14] Leo A Aroian and Howard Levene. The effectiveness of quality control charts. *Journal of the American Statistical Association*, 45(252):520–529, 1950.

[15] Parvin Asadzadeh, Rajkumar Buyya, C Ling Kei, Deepa Nayar, and Srikumar Venugopal. *Global grids and software toolkits: A study of four grid middleware technologies*. Wiley Press, New Jersey, USA, 2005.

[16] Ivan E Auger and Charles E Lawrence. Algorithms for the optimal identification of segment neighborhoods. *Bulletin of mathematical biology*, 51(1):39–54, 1989.

[17] Stefano Bagnasco, L Betev, P Buncic, F Carminati, C Cirstoiu, C Grigoras, A Hayrapetyan, A Harutyunyan, AJ Peters, and P Saiz. Alien: Alice environment on the grid. In *Journal of Physics: Conference Series*, volume 119, page 062012. IOP Publishing, 2008.

[18] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. Using magpie for request extraction and workload modelling. In *OSDI*, volume 4, pages 18–18, 2004.

[19] Daniel Barry and John A Hartigan. Product partition models for change point problems. *The Annals of Statistics*, pages 260–279, 1992.

[20] Daniel Barry and John A Hartigan. A bayesian analysis for change point problems. *Journal of the American Statistical Association*, 88(421):309–319, 1993.

[21] Michele Basseville and Igor V Nikiforov. Detection of abrupt changes: theory and applications. *Journal of the Royal Statistical Society-Series A Statistics in Society*, 158(1):185, 1995.

[22] Gowtham Bellala, Jason Stanley, Clayton Scott, and Suresh K Bhavnani. Active diagnosis via auc maximization: An efficient approach for multiple fault identification in large scale, noisy networks. *arXiv preprint arXiv:1202.3701*, 2012.

[23] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *NDSS*, 2011.

[24] Lucien Birgé and Pascal Massart. Minimal penalties for gaussian model selection. *Probability theory and related fields*, 138(1-2):33–73, 2007.

[25] H. Blodget. Amazon's cloud crash disaster permanently destroyed many customers' data, April 2011.

[26] B. Borchers. Csdp, a c library for semidefinite programming. *Optimization Methods and Software*, 11(1):613–623, 1999.

[27] Richard J Boys and Daniel A Henderson. A bayesian approach to dna sequence segmentation. *Biometrics*, 60(3):573–581, 2004.

[28] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

[29] Rasmus Bro. Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*, 38(2):149–171, 1997.

[30] Mark Brodie, Irina Rish, and Sheng Ma. Optimizing probe selection for fault localization. 2001.

[31] Mark Brodie, Irina Rish, and Sheng Ma. Intelligent probing: A cost-effective approach to fault diagnosis in computer networks. *IBM Systems Journal*, 41(3):372–385, 2002.

[32] Emmanuel J. Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Trans. Inf. Theor.*, 56(5):2053–2080, 2010.

[33] Jie Chen and A Arjun K Gupta. *Parametric Statistical Change Point Analysis: With Applications to Genetics, Medicine, and Finance*. Springer, 2012.

[34] Mike Y Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 595–604. IEEE, 2002.

[35] Betty HC Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al. *Software engineering for self-adaptive systems: A research roadmap*. Springer, 2009.

[36] Lu Cheng, Xuesong Qiu, Luoming Meng, Yan Qiao, and Raouf Boutaba. Efficient active probing for fault diagnosis in large scale and noisy networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[37] Walfredo Cirne, Francisco Brasileiro, Daniel Paranhos, Luís Fabrício W Góes, and William Voorsluys. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing*, 33(3):213–234, 2007.

[38] Ira Cohen, Jeffrey S Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, volume 4, pages 16–16, 2004.

[39] Les Cottrell, Warren Matthews, and Connie Logg. Tutorial on internet monitoring & pinger at slac, 2000.

[40] Matthew S Crouse, Robert D Nowak, and Richard G Baraniuk. Wavelet-based statistical signal processing using hidden markov models. *Signal Processing, IEEE Transactions on*, 46(4):886–902, 1998.

[41] R Ferreira Da Silva, T Glatard, F Desprez, et al. Self-healing of workflow activity incidents on distributed computing infrastructures. *Future Generation Computer Systems*, 2013.

[42] Scott W Davenport, Tarik Pertev Soydan, and Louis Yonlo Tsien. Method and system for platform independent fault management, May 31 2011. US Patent App. 13/118,723.

[43] Richard A Davis, Thomas C M Lee, and Gabriel A Rodriguez-Yam. Structural break estimation for nonstationary time series models. *Journal of the American Statistical Association*, 101(473):223–239, 2006.

[44] Frédéric Desobry, Manuel Davy, and Christian Doncarli. An online kernel change detection algorithm. *Signal Processing, IEEE Transactions on*, 53(8):2961–2974, 2005.

[45] Idris A Eckley, Paul Fearnhead, and Rebecca Killick. Analysis of changepoint models. 2011.

[46] Mattias Ellert, Michael Grønager, Aleksandr Konstantinov, Balázs Kónya, Jonas Lindemann, Ilja Livenson, Jakob Langgaard Nielsen, Marko Niinimäki, Oxana Smirnova, and Anders Wäänänen. Advanced resource connector middleware for lightweight computational grids. *Future Generation computer systems*, 23(2):219–240, 2007.

[47] Paul Fearnhead. Exact bayesian curve fitting and signal segmentation. *Signal Processing, IEEE Transactions on*, 53(6):2160–2166, 2005.

[48] Paul Fearnhead. Exact and efficient bayesian inference for multiple changepoint problems. *Statistics and computing*, 16(2):203–213, 2006.

[49] Paul Fearnhead and Zhen Liu. On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):589–605, 2007.

[50] Dawei Feng, Cécile Germain, and Tristan Glatard. Efficient distributed monitoring with active collaborative prediction. *Future Generation Computer Systems*, 29(8):2272–2283, 2013.

[51] Dawei Feng, Cecile Germain-Renaud, and Tristan Glatard. Distributed monitoring with collaborative prediction. In *12th Int. Symp. On Cluster, Cloud and Grid Computing*, pages 376–383, 2012.

[52] Rodrigo Fonseca, George Porter, Randy H Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, pages 20–20. USENIX Association, 2007.

[53] I. Foster. The globus toolkit for grid computing. In *IEEE Int. Symp. on Cluster Computing and the Grid*, 2001.

[54] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3):200–222, 2001.

[55] Edward B Fowlkes and Colin L Mallows. A method for comparing two hierarchical clusterings. *Journal of the American statistical association*, 78(383):553–569, 1983.

[56] A Frenkiel and H Lee. Epp: A framework for measuring the end-to-end performance of distributed applications. In *Proc. Performance Engineering Best Practices Conference*, number 1999, 1999.

[57] Garcıa-Neiva et al. A characterisation of dynamic distributed systems. Technical report, Institut Universitari Mixt Tecnologic deInformatica, 2012.

[58] Dennis Geels, Gautam Altekar, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Friday: Global comprehension for distributed replay. In *NSDI*, volume 7, pages 285–298, 2007.

[59] Dennis Michael Geels, Gautam Altekar, Scott Shenker, and Ion Stoica. *Replay debugging for distributed applications*, volume 68. 2006.

[60] Debanjan Ghosh, Raj Sharman, H Raghav Rao, and Shambhu Upadhyaya. Self-healing systemssurvey and synthesis. *Decision Support Systems*, 42(4):2164–2185, 2007.

[61] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[62] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, 1970.

[63] Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. *arXiv preprint arXiv:1302.7121*, 2013.

[64] Olivia A Grigg and David J Spiegelhalter. An empirical approximation to the null unbounded steady-state distribution of the cumulative sum statistic. *Technometrics*, 50(4):501–511, 2008.

[65] Peter D Grünwald, In Jae Myung, and Mark A Pitt. *Advances in minimum description length: Theory and applications*. MIT press, 2005.

[66] Boris Gruschke et al. Integrated event management: Event correlation using dependency graphs. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, pages 130–141, 1998.

[67] Xavier Guyon and Jian feng Yao. On the underfitting and overfitting sets of models chosen by order selection criteria. *Journal of Multivariate Analysis*, 70(2):221–249, 1999.

[68] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: Practical accountability for distributed systems. *ACM SIGOPS Operating Systems Review*, 41(6):175–188, 2007.

[69] Douglas M Hawkins. Testing a sequence of observations for a shift in location. *Journal of the American Statistical Association*, 72(357):180–186, 1977.

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

[71] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *22nd ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, 1999.

[72] Eric Hitti and Marie-Françoise Lucas. Wavelet-packet basis selection for abrupt changes detection in multicomponent signals. In *Proc. EUSIPCO*, pages 8–11, 1998.

[73] Holger Hoefling. A path algorithm for the fused lasso signal approximator. *Journal of Computational and Graphical Statistics*, 19(4):984–1006, 2010.

[74] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. In *NDSS*, 2008.

[75] Lajos Horváth. The maximum likelihood method for testing changes in the parameters of normal observations. *The Annals of statistics*, 21(2):671–680, 1993.

[76] Chengdu Huang, Ira Cohen, Julie Symons, and Tarek Abdelzaher. Achieving scalable automated diagnosis of distributed systems performance problems. *Enterprise Systems and Software Laboratory, HP Laboratories Palo Alto. Palo Alto, CA. Rep. HPL-2006-160*, 1, 2007.

[77] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[78] Tsuyoshi Ide and Hisashi Kashima. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 440–449. ACM, 2004.

[79] Carla Inclan and George C Tiao. Use of cumulative sums of squares for retrospective detection of changes of variance. *Journal of the American Statistical Association*, 89(427):913–923, 1994.

[80] Seidu Inusah and Tomasz J Kozubowski. A discrete analogue of the laplace distribution. *Journal of statistical planning and inference*, 136(3):1090–1102, 2006.

[81] Brad Jackson, Jeffrey D Scargle, David Barnes, Sundararajan Arabhi, Alina Alt, Peter Gioumousis, Elyus Gwin, Paungkaew Sangtrakulcharoen, Linda Tan, and Tun Tao Tsai. An algorithm for optimal partitioning of data on an interval. *Signal Processing Letters, IEEE*, 12(2):105–108, 2005.

[82] Nicholas A James and David S Matteson. ecp: An r package for nonparametric multiple change point analysis of multivariate data. Technical report, Technical report, Cornell University, 2013.

[83] Ravi Jhawar, Vincenzo Piuri, and Marco Santambrogio. Fault tolerance management in cloud computing: A system-level perspective. 2012.

[84] T. Joachims. A support vector method for multivariate performance measures. In *International Conference on Machine Learning (ICML)*, pages 377–384, 2005.

[85] Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. Detailed diagnosis in enterprise networks. *ACM SIGCOMM Computer Communication Review*, 39(4):243–254, 2009.

[86] Yoshinobu Kawahara and Masashi Sugiyama. Change-point detection in time-series data by direct density-ratio estimation. In *SDM*, volume 9, pages 389–400, 2009.

[87] Yoshinobu Kawahara, Takehisa Yairi, and Kazuo Machida. Change-point detection in time-series data based on subspace identification. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 559–564. IEEE, 2007.

[88] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[89] Charles Killian, James W Anderson, Ranjit Jhala, and Amin Vahdat. Life, death, and the critical transition: Finding liveness bugs in systems code. *NSDI 07: Networked Systems Design and Implementation*, pages 243–256, 2007.

[90] Rebecca Killick, Paul Fearnhead, and IA Eckley. Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598, 2012.

[91] Ross Kindermann, James Laurie Snell, et al. *Markov random fields and their applications*, volume 1. American Mathematical Society Providence, RI, 1980.

[92] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[93] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *14th ACM SIGKDD int. conf. on Knowledge discovery and data mining*, pages 426–434, 2008.

[94] Yehuda Koren. Collaborative Filtering with Temporal Dynamics. In *15th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 447–456, 2009.

[95] Yehuda Koren. The BellKor Solution to the Netflix Grand Prize. Technical report, Yahoo! Research, 2009.

[96] Akshay Krishnamurthy and Aarti Singh. Low-rank matrix and tensor completion via adaptive sampling. In *Advances in Neural Information Processing Systems*, pages 836–844, 2013.

[97] Rupa Krishnan, Harsha V Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize cdn performance. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 190–201. ACM, 2009.

[98] Erwin Laure, A Edlund, F Pacini, P Buncic, M Barroso, A Di Meglio, F Prelz, A Frohner, O Mulmo, A Krenek, et al. Programming the grid with glite. Technical report, 2006.

[99] Helene Laurent and Christian Doncarli. Stationarity index for abrupt changes detection in the time-frequency plane. *Signal Processing Letters, IEEE*, 5(2):43–45, 1998.

[100] Jason Lee, Ben Recht, Nathan Srebro, Joel Tropp, and Ruslan Salakhutdinov. Practical large-scale optimization for max-norm regularization. In *Advances in Neural Information Processing Systems*, pages 1297–1305, 2010.

[101] Bin Li, Xingquan Zhu, Ruijiang Li, Chengqi Zhang, Xiangyang Xue, and Xindong Wu. Cross-domain collaborative filtering over time. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, pages 2293–2298. AAAI Press, 2011.

[102] Zhichun Li, Ming Zhang, Zhaosheng Zhu, Yan Chen, Albert G Greenberg, and Yi-Min Wang. Webprophet: Automating performance prediction for web services. In *NSDI*, volume 10, pages 10–10, 2010.

[103] Yi Lin, Grace Wahba, Hao Zhang, and Yoonkyung Lee. Statistical properties and adaptive tuning of support vector machines. *Machine Learning*, 48:115–136, September 2002.

[104] Diane Lingrand, Johan Montagnat, Janusz Martyniak, and David Colling. Optimization of jobs submission on the egee production grid: modeling faults using workload. *Journal of Grid Computing*, 8(2):305–321, 2010.

[105] Max A Little and Nick S Jones. Generalized methods and solvers for noise removal from piecewise constant signals. i. background theory. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 467(2135):3088–3114, 2011.

[106] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye. Tensor completion for estimating missing values in visual data. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2114–2121. IEEE, 2009.

[107] Jun S. Liu and Charles E. Lawrence. Bayesian inference on biopolymer models. *Bioinformatics*, 15(1):38–52, 1999.

[108] Xuezheng Liu, Zhenyu Guo, Xi Wang, Feibo Chen, Xiaochen Lian, Jian Tang, Ming Wu, M Frans Kaashoek, and Zheng Zhang. D3s: Debugging deployed distributed systems. In *NSDI*, volume 8, pages 423–437, 2008.

[109] Xuezheng Liu, Wei Lin, Aimin Pan, and Zheng Zhang. Wids checker: Combating bugs in distributed systems. In *NSDI*, 2007.

[110] J. Löfberg. Yalmip : A toolbox for modeling and optimization in MATLAB. 2004.

[111] Bruce B Lowekamp. Combining active and passive network measurements to build scalable monitoring systems on the grid. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):19–26, 2003.

[112] James M Lucas and Michael S Saccucci. Exponentially weighted moving average control schemes: properties and enhancements. *Technometrics*, 32(1):1–12, 1990.

[113] Lester W. Mackey, David Weiss, and Michael I. Jordan. Mixed membership matrix factorization. In *27th International Conference on Machine Learning (ICML-10)*, 2010.

[114] T Maeno. Panda: distributed production and distributed analysis system for atlas. *Journal of Physics: Conference Series*, 119(6):062036, 2008.

[115] Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[116] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[117] David S Matteson and Nicholas A James. A nonparametric approach for multiple change point analysis of multivariate data. *arXiv preprint arXiv:1306.4933*, 2013.

[118] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[119] Katharina Morik, Peter Brockhausen, and Thorsten Joachims. Combining statistical learning with a knowledge-based approach - a case study in intensive care monitoring. In *16th Int. Conf. on Machine Learning*, pages 268–277, 1999.

[120] J. T. Moscicki. Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data. In *Nuclear Science Symposium Conference Record, 2003 IEEE*, volume 3, pages 1617–1620 Vol.3, 2003.

[121] Valentina Moskvina and Anatoly Zhigljavsky. An algorithm based on singular spectrum analysis for change-point detection. *Communications in Statistics-Simulation and Computation*, 32(2):319–352, 2003.

[122] Adam B Olshen, ES Venkatraman, Robert Lucito, and Michael Wigler. Circular binary segmentation for the analysis of array-based dna copy number data. *Biostatistics*, 5(4):557–572, 2004.

[123] AN Pettitt. A non-parametric approach to the change-point problem. *Applied statistics*, pages 126–135, 1979.

[124] Franck Picard, Stephane Robin, Marc Lavielle, Christian Vaisse, and Jean-Jacques Daudin. A statistical approach for array cgh data analysis. *BMC bioinformatics*, 6(1):27, 2005.

[125] Aleksey S Polunchenko and Alexander G Tartakovsky. State-of-the-art in sequential change-point detection. *Methodology and Computing in Applied Probability*, 14(3):649–684, 2012.

[126] Ian Porteous, Evgeniy Bart, and Max Welling. Multi-hdp: a non parametric bayesian model for tensor factorization. In *23rd Conf. on Artificial Intelligence*, pages 1487–1490, 2008.

[127] Mohammed Abdul Qadeer, Mohammad Zahid, Arshad Iqbal, and MisbahurRahman Siddiqui. Network traffic analysis and intrusion detection using packet sniffer. In *Communication Software and Networks, 2010. ICCSN'10. Second International Conference on*, pages 313–317. IEEE, 2010.

[128] Lawrence Rabiner and B Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[129] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[130] Benjamin Recht. A simpler approach to matrix completion. *J. Mach. Learn. Res.*, 12:3413–3430, 2011.

[131] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Rev.*, 52(3):471–501, 2010.

[132] Jasson DM Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[133] Patrick Reynolds, Charles Edwin Killian, Janet L Wiener, Jeffrey C Mogul, Mehul A Shah, and Amin Vahdat. Pip: Detecting the unexpected in distributed systems. In *NSDI*, volume 6, pages 115–128, 2006.

[134] H. Rifai, S. Mohammed, and A. Mellouk. A brief synthesis of QoS-QoE methodologies. In *10th Int. Symp. on Programming and Systems*, pages 32–38, 2011.

[135] Guillem Rigaill. Pruned dynamic programming for optimal multiple change-point detection. *arXiv preprint arXiv:1004.0887*, 2010.

[136] Irina Rish and al. al. al. al. Adaptive diagnosis in distributed systems. *IEEE Trans. Neural Networks*, 16(5):1088–1109, 2005.

[137] Irina Rish, Mark Brodie, Natalia Odintsova, Sheng Ma, and Genady Grabarnik. Real-time problem determination in distributed systems using active probing. In *Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP*, volume 1, pages 133–146. IEEE, 2004.

[138] Irina Rish and Gerald Tesauro. Estimating end-to-end performance by collaborative prediction with active sampling. In *Integrated Network Management*, pages 294–303, 2007.

[139] Maria L Rizzo and Gábor J Székely. Disco analysis: A nonparametric extension of analysis of variance. *The Annals of Applied Statistics*, 4(2):1034–1055, 2010.

[140] Gordon J Ross. Parametric and nonparametric sequential change detection in r: The cpm package. *Journal of Statistical Software*, 2013.

[141] Saharon Rosset and Ji Zhu. Piecewise linear regularized solution paths. *The Annals of Statistics*, pages 1012–1030, 2007.

[142] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[143] Ramendra K Sahoo, Adam J Oliner, Irina Rish, Manish Gupta, José E Moreira, Sheng Ma, Ricardo Vilalta, and Anand Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435. ACM, 2003.

[144] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.

[145] Ruslan Salakhutdinov and Nathan Srebro. Collaborative filtering in a non-uniform world: Learning with the weighted trace norm. In *24th Conference on Neural Information Processing Systems (NIPS)*, pages 2056–2064, 2010.

[146] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10, 2010.

[147] Judith Samson and Manfred Warmuth. Predicting round trip time for the tcp protocol. 2013.

[148] Gideon Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[149] Mouhamadou Seck, Ivan Magrin-Chagnolleau, and Frédéric Bimbot. Experiments on speech tracking in audio documents using gaussian mixture modeling. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 1, pages 601–604. IEEE, 2001.

[150] Beilu Shao, Daniele Renzi, Peter Amon, Georgios Xilouris, Nikos Zotos, Stefano Battista, Anastasios Kourtis, and Marco Mattavelli. An adaptive system for real-time scalable video streaming with end-to-end qos control. In *Image Analysis for Multimedia Interactive Services (WIAMIS), 2010 11th International Workshop on*, pages 1–4. IEEE, 2010.

[151] Atul Singh, Petros Maniatis, Timothy Roscoe, and Peter Druschel. Using queries for distributed monitoring and forensics. *ACM SIGOPS Operating Systems Review*, 40(4):389–402, 2006.

[152] Nathan Srebro, Jason D. M. Rennie, and Tommi S. Jaakola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems 17*, pages 1329–1336, 2005.

[153] Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In *Learning Theory*, pages 545–560. Springer, 2005.

[154] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[155] Gabor J Szekely and Maria L Rizzo. Hierarchical clustering via joint between-within distances: Extending ward's minimum variance method. *Journal of classification*, 22(2):151–183, 2005.

[156] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research (JMLR)*, 10:623–656, 2009.

[157] Ryan Joseph Tibshirani. *The solution path of the generalized lasso*. Stanford University, 2011.

[158] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2, March 2002.

[159] Hai Tran and Abdelhamid Mellouk. Qoe model driven for network services. In *Wired/Wireless Internet Communications*, volume 6074 of *LNCS*, pages 264–277. 2010.

[160] A Tsaregorodtsev and al. al. al. al. DIRAC3 . The New Generation of the LHCb Grid Software. *Journal of Physics: Conference Series*, 219(6):062029, 2009.

[161] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, September 2005.

[162] Ryan Darby Turner. Gaussian processes for state space models and change point detection. 2012.

[163] Tim van Erven, Peter Grünwald, and Steven de Rooij. Catching up faster by switching sooner: a predictive approach to adaptive estimation with an application to the aic–bic dilemma. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(3):361–417, 2012.

[164] Weijia Wang and Michèle Sebag. Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search. *Machine learning*, 92(2-3):403–429, 2013.

[165] Wenwei Weng, Minghui Zhu, and Vishnu Kant Varma. Auto probing endpoints for performance and fault management, November 19 2010. US Patent App. 12/950,916.

[166] Keith Winstein and Hari Balakrishnan. End-to-end transmission control by modeling uncertainty about the network state. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 19. ACM, 2011.

[167] KJ Worsley. An improved bonferroni inequality and applications. *Biometrika*, 69(2):297–302, 1982.

[168] Xiang Xuan and Kevin Murphy. Modeling changing dependency structure in multivariate time series. In *Proceedings of the 24th international conference on Machine learning*, pages 1055–1062. ACM, 2007.

[169] Lian Yan, Robert H. Dodier, Michael Mozer, and Richard H. Wolniewicz. Optimizing Classifier Performance via an Approximation to the Wilcoxon-Mann-Whitney Statistic. In *20th International Conference on Machine Learning (ICML-03)*, pages 848–855, 2003.

[170] Yi-Ching Yao. Estimation of a noisy discrete-time step function: Bayes and empirical bayes approaches. *The Annals of Statistics*, pages 1434–1447, 1984.

[171] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 1(1):6, 2007.

[172] Ding Yuan, Haohui Mai, Weiwei Xiong, Lin Tan, Yuanyuan Zhou, and Shankar Pasupathy. Sherlog: error diagnosis by connecting clues from run-time logs. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 143–154. ACM, 2010.

[173] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163–188, 2005.

[174] Nancy R Zhang and David O Siegmund. A modified bayes information criterion with applications to the analysis of comparative genomic hybridization data. *Biometrics*, 63(1):22–32, 2007.

[175] Xiangliang Zhang, Cyril Furtlehner, Cecile Germain-Renaud, and Michèle Sebag. Data stream clustering with affinity propagation. *IEEE Transactions on Knowledge and Data Engineering*, 2014.

[176] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3):270–284, 2011.

[177] Alice X Zheng, Irina Rish, and Alina Beygelzimer. Efficient test selection in active diagnosis via entropy approximation. *arXiv preprint arXiv:1207.1418*, 2012.

[178] Zibin Zheng and Michael R. Lyu. Collaborative reliability prediction of service-oriented systems. In *32nd ACM/IEEE Int. Conf. on Software Engineering*, 2010.

[179] Ziming Zheng, Zhiling Lan, Byung-Hoon Park, and Al Geist. System log pre-processing to improve failure prediction. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 572–577. IEEE, 2009.

[180] Wenchao Zhou. Fault management in distributed systems. 2010.