

Mining Logs Files for Data-Driven System Management

Wei Peng
School of Computer Science
Florida International University
Miami, FL 33199
wpeng002@cs.fiu.edu

Tao Li
School of Computer Science
Florida International University
Miami, FL 33199
taoli@cs.fiu.edu

Sheng Ma
Machine Learning for Systems
IBM T.J. Watson Research
Center
Hawthorne, NY 10532
shengma@us.ibm.com

ABSTRACT

With advancement in science and technology, computing systems are becoming increasingly more complex with an increasing variety of heterogeneous software and hardware components. They are thus becoming increasingly more difficult to monitor, manage and maintain. Traditional approaches to system management have been largely based on domain experts through a knowledge acquisition process that translates domain knowledge into operating rules and policies. This has been well known and experienced as a cumbersome, labor intensive, and error prone process. In addition, this process is difficult to keep up with the rapidly changing environments. There is thus a pressing need for automatic and efficient approaches to monitor and manage complex computing systems.

A popular approach to system management is based on analyzing system log files. However, some new aspects of the log files have been less emphasized in existing methods from data mining and machine learning community. The various formats and relatively short text messages of log files, and temporal characteristics in data representation pose new challenges. In this paper, we will describe our research efforts on mining system log files for automatic management. In particular, we apply text mining techniques to categorize messages in log files into common situations, improve categorization accuracy by considering the temporal characteristics of log messages, and utilize visualization tools to evaluate and validate the interesting temporal patterns for system management.

Keywords

System Log, Categorization, Temporal Information, Visualization, Naive Bayes, Hidden Markov Model

1. INTRODUCTION

When problems occur, traditional approaches for trouble shooting rely on the knowledge and experience of domain experts to figure out ways to discover the rules or look for the problem solutions laboriously. It has been estimated that, in medium and large companies, anywhere from 30% to 70% of their information technology resources are used in dealing with problems [22]. It is unrealistic and inefficient to depend on domain experts to manually deal with complex problems in ever-changing computing systems.

Modern computing systems are instrumented to generate huge amounts of system log data. The data in the log files describe the status of each component and record system operational changes, such as the starting and stopping of services, detection of network applications, software configuration modifications, and software

execution errors. Analyzing log files, as an attractive approach for automatic system management and monitoring, has been enjoying a growing amount of attention. However, several new aspects of the system log data have been less emphasized in existing analysis methods from data mining and machine learning community and pose several challenges calling for more research. The aspects include disparate formats and relatively short text messages in data reporting, asynchronism in data collection, and temporal characteristics in data representation.

First, the heterogeneous nature of the system makes the data more complex and complicated [10]. As we know, a typical computing system contains different devices (e.g., routers, processors, and adapters) with different software components (e.g., operating systems, middleware, and user applications), possibly from different providers (e.g., Cisco, IBM, and Microsoft). These various components have multiple ways to report events, conditions, errors and alerts. The heterogeneity and inconsistency of log formats make it difficult to automate problem determination. For example, there are many different ways for the components to report the start up process. Some might log “the component has started”, while others might say that “the component has changed the state from starting to running”. This makes it difficult to perform automated analysis of the historical event data across multiple components when problems occur as one need to know all the messages that reflect the same status, for all the components involved in the solution [26]. To enable automated analysis of the historical event data across multiple components, we need to categorize the text messages with disparate formats into common situations. Second, text messages in the log files are relatively short with a large vocabulary size [25]. Hence, care must be taken when applying traditional document processing techniques. Third, each text message usually contains a timestamp. The temporal characteristics provide additional context information of the messages and can be used to facilitate data analysis.

In this paper, we describe our research efforts to address the above challenges in mining system logs. In particular, we propose to mine system log files for computing system management by acquiring the needed knowledge automatically from a large amount of historical log data, possibly from different types of information sources such as system errors, resource performance metrics, and trouble ticket text records. Specifically, we will apply text mining techniques to automatically categorize the text messages into a set of common categories, incorporate temporal information to improve categorization performance, and utilize visualization tools to evaluate and validate the interesting temporal patterns for system management. A preliminary version of this paper has been presented as a short paper [20] at The 2nd IEEE International Conference on Autonomic Computing (ICAC-05)

It should be noted that our framework is complementary to the current knowledge-based approaches, which are based on elicitation of knowledge from domain experts. Automated log data analysis can be performed without much domain knowledge and its results provide guidance for network managers to perform their jobs more effectively. Moreover, the available domain knowledge can be used to validate, improve, and refine data analysis.

The rest of the paper is organized as follows: Section 2 applies text mining techniques to categorize text messages into a set of common categories, Section 3 proposes two approaches of incorporating temporal information to improve the categorization performance, Section 4 discusses visualization that can be used to discover, interpret and validate the temporal patterns/relationships for system management, Section 5 presents our experimental results, and finally Section 6 provides conclusions and discussions.

2. SYSTEM LOG CATEGORIZATION

2.1 Common Categories

The disparate logging mechanisms impede problem investigation because of no standard approach for classifying them [26]. Two components experiencing the same problem may use different formats to report the same information. For instance, when component A starts running, it reports “A has started” in log file. However, when component B starts running, it may report “B has begun execution”. Although both messages have the same content meaning, they are reported differently. This makes it difficult to correlate logs from across different components when problems occur.

In order to create consistency across similar fields and improve the ability to correlate across multiple logs, it is necessary to transform the messages in the log files into a common base [3] (i.e., a set of common categories). In this paper, we first manually determine a set of categories as the basis for transformation. The set of categories is based on the CBE (Common Base Event) format established by IBM initiative [26]. CBE provides a finite set of canonical situations after analyzing thousands of log entries across multiple IBM and non-IBM products. Although we use CBE as an internal presentation here, the problem and the proposed approach are generic and can be extended for other data formats as well.

Specifically, the set of categories¹ includes *start*, *stop*, *dependency*, *create*, *connection*, *report*, *request*, *configuration*, and *other*. The category *start* deals with the start-up process of the component. The category *stop* deals with the exit process. The category *dependency* handles messages which report components cannot find some features or other components. The category *create* handles the creation problems of components. The category *connection* is used to identify aspects about a connection to another component. The category *report* deals with performance, task-relative reporting messages. The category *request* identifies the completion status of a request. The category *other* identifies the blank messages or messages having vague meanings.

2.2 Message Categorization

Given the set of common categories, we can then categorize the messages reported by different components into the prescribed categories. This can be viewed as a text categorization problem where the goal is to assign predefined category labels to unlabeled documents based on the likelihood inferred from the training set of labeled documents [9; 24].

¹Our future work includes exploring more categories as well as investigating approaches to infer categories from the log data automatically.

In our work, we use naive Bayes as our classification approach as it is among the most successful known algorithms for learning in text categorization [23]. System log files usually have a large size of vocabulary and most of the system log messages contain a free-format 1024-byte ASCII description of the event [25]. Naive Bayes is a simple practical generative classification algorithm based on Bayes Theorem with the assumption of class conditional independence. Basically it assumes that the text data is generated by a parametric model, and uses training data to calculate Bayes-optimal estimates of the model parameters [17]. Then, it classifies test data using the generative model by calculating the posterior probability that a class would have generated the test data in question. The most probable class is then assigned to the test data [19].

2.3 Naive Bayes Classifier

Formally, suppose there are L categories, denoted by C_1, C_2, \dots, C_L , and each category is generated from some probability distribution. Each document d_i is created by first selecting a category according to the prior $P(C_j)$ followed by the generation according to the distribution of the chosen category $P(d_i|C_j)$. We can characterize the likelihood of a document with the sum of probability over all the categories

$$P(d_i) = \sum_{j=1}^L P(C_j)P(d_i|C_j).$$

Given a set of training samples S , the naive Bayes classifier uses S to estimate $P(d_i|C_j)$ and $P(C_j)$. The estimated probabilities are typically calculated in a straightforward way from training data. To classify a new sample, it uses Bayes rule to compute class posterior

$$P(C_j|d_i) = \frac{P(C_j)P(d_i|C_j)}{\sum_{j=1}^L P(C_j)P(d_i|C_j)}.$$

The predicted class for the document d_i is then just $\text{argmax}_j P(C_j|d_i)$. In our work, we use multinomial probability model for document generation of each category. More details on Naive Bayes can be found in [19].

3. INCORPORATING THE TEMPORAL INFORMATION

The classification performance achieved by the Naive Bayes classifier in our practice is not satisfactory as will be shown in Section 5. In many scenarios, text messages generated in the log files usually contain timestamps. The temporal characteristics provide additional context information of the messages and can be used to facilitate data analysis. As an example, knowing the current status of a network component would help forecast the possible types of messages it will generate. These structural constraints can be naturally represented using naive Bayes algorithm and hidden Markov models [11]. In this section, we describe two approaches of utilizing the temporal information to improve classification performance. Our experiments in Section 5 show that both approaches greatly improve the categorization accuracy.

3.1 Modified Naive Bayes algorithm

To facilitate the discussion, Table 1 gives an example of the system log files extracted from a windows XP machine². In the example, four columns, i.e., *Time*, *EventSource*, *Msg* and *State* are listed. *Time* represents the generated timestamp of log messages, *EventSource* identifies the component which reports the message,

²The details of the data collection will be described in Section 5.

Time	EventSource	msg	State
1093359583	UPHClean	The following handles in user profile hive FIU-SCS \ lcruz01 (S-1-5-21-876885435-739206947-926709054-2036) have been closed because they were preventing the profile from unloading successfully wmpvse.exe (2432)HKCU (0x228)	report
1093359583	UPHClean	User profile hive cleanup service version 1.5.5.21 started successfully.	start
1093359603	AutoEnrollment	Automatic certificate enrollment for local system failed to contact the active directory (0x8007054b). The specified domain either does not exist or could not be contacted.Enrollment will not be performed. 	dependency
1093359605	Inventory Scanner	Unable to open output file C: \invdelta.tmp.	dependency
1093359606	Inventory Scanner	LDISCN32: Can't create temporary file.	create
1093359641	Inventory Scanner	Unable to open output file C: \invdelta.tmp.	dependency
1093359918	Inventory Scanner	LDISCN32: Can't create temporary file.	create
1093359583	Userenv		other
1093359584	MsiInstaller	Product: WebFldrs XP – Configuration completed successfully. 	dependency

Table 1: An example fraction of a log file with the manual label of each message

Msg lists the content of text message, and *State* labels the current category.

If a sequence of log messages are considered, the accuracy of categorization for each message can be improved as the structure relationships among the messages can be used for analysis. For example, the components usually first start up a process, then stop the process at a later time. That is, the *stop* message usually occurs after the *start* message. For another example, as shown in Table 1, the component *Inventory Scanner* will report *dependency* messages if it cannot find some features or components. This happens especially just right after it generates the *create* message– “cannot create *” (* is the abbreviation of arbitrary words. It represents certain components or features here). Thus it is beneficial to take the temporal information into consideration.

In order to take the relationships between adjacent messages into account, we make some modifications to the naive Bayes algorithm. Suppose we are given a sequence of adjacent messages $D = (d_1, d_2, \dots, d_T)$. let Q_i be the category labels for message d_i (i.e., Q_i is one of C_1, C_2, \dots, C_L). Now we want to classify d_{i+1} . Note that

$$\begin{aligned} P(Q_{i+1}|d_{i+1}, Q_i) &= \frac{P(Q_{i+1}, d_{i+1}|Q_i)}{P(d_{i+1}|Q_i)} \\ &= \frac{P(Q_{i+1}|Q_i)P(d_{i+1}|Q_{i+1}, Q_i)}{P(d_{i+1}|Q_i)} \end{aligned}$$

Assuming that d_{i+1} is conditionally independent of Q_i given Q_{i+1} , $P(d_{i+1}|Q_{i+1}, Q_i) = P(d_{i+1}|Q_{i+1})$. In addition, once Q_i is determined, $P(d_{i+1}|Q_i)$ is then fixed. Hence maximizing $P(Q_{i+1}|d_{i+1}, Q_i)$ is equivalent to maximizing $P(Q_{i+1}|Q_i)P(d_{i+1}|Q_{i+1})$. Observe that $P(d_{i+1}|Q_{i+1}) = \frac{P(Q_{i+1}|d_{i+1})P(d_{i+1})}{P(Q_{i+1})}$. If we assume a uniform prior on $P(Q_{i+1})$, in order to maximize $P(d_{i+1}|Q_{i+1})$, it is enough to maximize $P(Q_{i+1}|d_{i+1})$. Therefore, in summary, we assign d_{i+1} to the category Q_{i+1} which is

$$\operatorname{argmax}_j (P(C_j|d_{i+1}) \times P(C_j|Q_i)).$$

In other words, we aim at maximizing the multiplication of text classification probability $P(C_j|d_{i+1})$ and state transition probability $P(C_j|Q_i)$. The transition probability $P(C_j|Q_i)$ can be estimated from training log files.

3.2 Hidden Markov Model

Hidden Markov Model(HMM) is another approach to incorporate the temporal information for message categorization. The temporal relations among messages are naturally captured in HMM [13]. The Hidden Markov Model is a finite set of states, each of which is associated with a (generally multidimensional) probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities [21]. In a particular state an outcome or observation can be generated, according to the associated probability distribution. The model describes a probabilistic generative process whereby a sequence of symbols is produced by starting in some state, transitioning to a new state, emitting a symbol selected by that state, transitioning again, emitting another symbol and so on until a designated final state is reached. Associated with each of a set of states, $S = \{s_1, \dots, s_n\}$, is a probability distribution over the symbols in the emission vocabulary $K = \{k_1, \dots, k_m\}$. The probability that state s_j will emit the vocabulary item k is written $P(k|s_j)$. Similarly, associated with each state is a distribution over its outgoing transitions. The probability of moving from state s_i to state s_j is written $P(s_j|s_i)$. There is also a prior state distribution $\pi(s)$. Training data consists of several sequences of observed emissions, one of which would be written $\{o_1, \dots, o_x\}$ [5].

In our experiment, the category labels we specified in the above sections are regarded as states. The emitting observation symbols are the log messages corresponding to their state labels. HMM explicitly considers the state transition sequence. It can also be used to compare all state paths from the start state to the destination state, and then choose the best state sequence that emits a certain observation sequence. So it works well in labeling continuous log messages. When one log message has been assigned several competitive state labels by text classification, HMM selects a certain label by traversing the best state sequence. The state transition probability is calculated from the training log data sets, for example, in our experiment the probability of state *create* to state *dependency* is 0.4470. The probability of emitting messages can be estimated as the ratio of the occurrence probabilities of log messages to the occurrence of their states in the training data. Viterbi algorithm is used to find the most possible state sequence that emits the given log messages, that is, finding the most possible state labels to assign to these log messages [12].

The advantage of HMM lies in the fact that it calculates all possible state paths from the beginning state to the current state. As for some messages that have several competitive state labels, the text classification probability and the occurrence probability are combined to calculate the observation probability. For example, the message “The Windows Installer initiated a system restart to complete or continue the configuration of ‘Microsoft XML Parser’.” has 0.4327 probability to be categorized into *configuration* state, and has 0.4164 probability to be labeled as *stop* state. This means that this message can be possibly emitted by two states. In this case, the observation probability not only needs to consider the occurrence probability, but also the text classification probability.

4. VISUALIZE EVENT RELATIONSHIPS

Once the messages in the log file are transformed into common categories, it is then possible to analyze the historical event data across multiple components to discover interesting patterns embedded in the data. Each message in log files usually has a timestamp and we will try to discover the interesting temporal patterns. Temporal patterns of interest appear naturally in the system management applications [7]. Specifically, a computer system problem may trigger a series of symptom events/activities. Such a sequence of symptom events provides a natural signature for identifying the root cause. For example, when component A reports “A is stopped”, how should we respond to this *stop* message? Is this a failure in the component A or just a natural exit after successful completion of tasks? In fact, the event “A is stopped” does not necessarily indicate that an error or a failure happened. We noticed that in our log data files, the *stop* message often occurs after the component reports *dependency* message (i.e., it cannot find certain features or components). In this case, the *stop* message signifies the failure of a task execution. However, if there is no *dependency* message before the *stop* message, the *stop* message usually indicates the completion of a task. Thus to decide the proper actions in response to the *stop* messages, we need to check what happens right before them. Thus knowing the temporal patterns can help us pinpoint the root cause and take proper actions.

Visualization can help the users understand and interpret patterns in the data [6; 27]. For example, a scatter plot can help network operators to identify patterns of significance from a large amount of monitoring data [4]. Once we categorize each message into a set of categories (or event types), each message can then be represented as a three-tuple $\langle s, t, c \rangle$ where s is the category label or event type, t is the timestamp, and c is the source component of this log message. We can use visualization tools to discover, interpret and validate interesting patterns. We will describe our visualization effort in Section 5.

5. EXPERIMENTS

In this section, we present our experimental results. Section 5.1 describes the log data generation, Section 5.2 gives the experimental results using Naive Bayes, Section 5.3 shows the results using the two approaches for incorporating temporal information, Section 5.4 illustrates our visualization efforts on the log files.

5.1 Log Data Generation

The log files used in our experiments are collected from several different machines with different operating systems in the School of Computer Science at Florida International University. We use logdump2td (NT data collection tool) developed by Event Mining Team at IBM T.J. Watson research center. The raw log files has 14 columns, which are sequence, time, EventType, ReportTime, Log-

TypeName, LogTypeID, EventSourceName, EventSource, HostName, HostID, severity, Category, and Msg. To preprocess text messages, we remove stop words and skip html labels. We use accuracy, defined as the proportion of messages that are correctly assigned to a category, to evaluate the classification performance and use the random 30-70 data split for training and testing.

5.2 Log Categorization Using Naive Bayes

Our first experiment is on using naive Bayes classifier to categorize log messages. The classification tool is built on the Bow (A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering) software from Carnegie Mellon University³. The training data are labeled with nine categories, i.e., *configuration*, *connection*, *create*, *dependency*, *other*, *report*, *request*, *start*, and *stop*. The accuracy of log categorization is about 0.7170.

Table 2 lists the keywords and their probabilities in the corresponding classes. These keywords have high occurrence probability and information gain associated with a category and they can be used to describe the category.

As we know, one difficulty for classifying log messages with various formats is that different platforms (e.g., Windows and Linux) usually have different set of terminologies. The classifier model built from Windows Log files may not be able to achieve good performance on Linux log files due to the differences in vocabulary. One of our future tasks is to map the keywords between different platforms.

5.3 Incorporating Temporal Information

5.3.1 Modified Naive Bayes

In order to improve the classification accuracy of log messages, we consider the time stamp of log messages and adjacent state transition as discussed in Section 3.1. Table 3 lists the state transition probability $P(C_j|C_i)$ derived from training data. The leftmost column lists previous category labels C_i . The top row lists current category labels C_j . Every entry contains a transition probability from a row category to the corresponding column category.

After considering the state transition probability, the categorization accuracy is improved to 0.8232. We describe three examples (shown in Table 4, Table 5, Table 6) taken from our experiments to illustrate the effectiveness of our approach. The columns in the three tables are, from left to right, time stamps, category labels obtained by incorporating transition probability (denoted as *TC*), category label without considering the transition probability (denoted as *Naive*), the reporting component, and the log message. In each table, the first row represents the previous message, and the second row represents the current message. By incorporating the transition probability, the category of the previous message contributes to the decision on the category of the current message.

In Table 7, each row corresponds to one current log message from the above three examples (the leftmost column identifies the time stamp of these messages). Each entry is the text classification probability $Pr(C_j|d_i)$, the probability of row message categorized into the column category. We can observe the labels obtained by naive Bayes text classification on these three example messages are not correct, while the modified approach produces the correct label. For instance, in the first example, the transition probability from *configuration* to *start* is 0.1869, the probability from *configuration* to *configuration* is 0.1111. Naive Bayes classification assigns this message to *start* with probability of 0.4430199. It assigns this

³The tool can be downloaded at <http://www-2.cs.cmu.edu/~mccallum/bow/>.

Configuration:	0.140 product, 0.143 configuration, 0.133 completed
Connection:	0.073 inventory, 0.073 server, 0.073 respond, 0.038 network, 0.038 connection, 0.050 party, 0.050 root
Create:	0.215 create, 0.215 temporary, 0.215 file
Dependency:	0.063 exist, 0.063 directory, 0.063 domain, 0.063 contacted, 0.063 contact, 0.063 failed, 0.063 certificate, 0.127 enrollment
Report:	0.084 profile, 0.059 service, 0.097 version, 0.097 faulting, 0.050 application, 0.050 module 0.053 fault, 0.050 address
Request:	0.083 completed, 0.060 update, 0.097 installation
Start:	0.150 service, 0.190 started, 0.088 application, 0.037 starting
Stop:	0.059 stopped, 0.054 restarted, 0.011 completed, 0.032 failed, 0.054 shell, 0.054 explorer

Table 2: Keywords and their probabilities

	configuration	connection	create	dependency	other	report	request	start	stop
configuration	0.1111	0.1111	0	0.2359	0.2012	0.0171	0.0085	0.1869	0.1282
connection	0	0.2889	0	0.2966	0.2608	0.0821	0.0201	0.0390	0.0125
create	0.0043	0	0	0.4470	0.4550	0.0461	0.0101	0.0231	0.0144
dependency	0.0023	0.0036	0.2238	0.4024	0.3078	0.0265	0.0011	0.0279	0.0047
other	0.0530	0.2119	0.0115	0.1775	0.2765	0.0354	0	0.2343	0
report	0.0427	0	0	0.2012	0.2204	0.3310	0.0869	0.0875	0.0304
request	0	0.0725	0	0.1500	0.0725	0.0474	0.4317	0.0784	0.1474
start	0	0.0366	0	0.3737	0.3428	0	0.0285	0.2143	0.0041
stop	0.1429	0.0952	0	0.2646	0.1455	0.0741	0	0.0476	0.2302

Table 3: The matrix table of state transition probability

message to *configuration* category with probability 0.4981456. By incorporating temporal information, our modified naive Bayes algorithm assign this message to *start* as $0.1869 \times 0.4430199 > 0.1111 \times 0.4981456$. Our experiments show that the modified algorithm improves the accuracy of log categorization from 0.7170 to 0.8232.

5.3.2 HMM

Our implementation of HMM tool is based on the package UMDHMM version 1.02 from University of Maryland⁴. The state transition probability is calculated from the training log data sets and the probability of emitting messages can be estimated as the ratio of the occurrence probabilities of log messages to the occurrence of their states in the training model. We use Baum-Welch procedure to learn the HMM parameters [2]. Viterbi algorithm can then be used to discover the best state sequence. Because of the large number of emitting messages, the observation probability table is omitted here. The accuracy of using HMM categorization is 0.85.

5.3.3 Summary

The experimental results on message categorization are summarized in Table 8. In summary, our experiments show that both the modified Bayes algorithm and HMM enhance the classification accuracy and HMM achieves the best performance in our practice. However, constructing HMM requires more computational costs.

⁴The package can be downloaded at <http://www.cfar.umd.edu/~kanungo/software/software.html>.

Intuitively, the modified Bayes algorithm considers only adjacent state transitions while HMM considers all possible state paths. The choice between the two approaches for incorporating temporal information is problem-dependent in practice.

Methods	Accuracy
Naive Bayes	0.7170
Modified Naive Bayes	0.8232
HMM	0.85

Table 8: Performance Comparisons on Message Categorization

5.4 Visualize Event Relationships

Once we categorize each message into a set of categories (or event types), we can use visualization tools to discover, interpret and validate interesting patterns. Visualization can help the users understand and interpret patterns in the data [6; 27]. For example, a scatter plot can help network operators to identify patterns of significance from a large amount of monitoring data [4]. We perform visualization using the EventBrowser [15]. The visualization of events makes it convenient for people to find interesting temporal patterns. The temporal pattern found and validated can be used to provide better mechanisms for system management, especially when complex integrations are typically hard for humans to capture.

Figure 1 is derived from a desktop machine named “raven” in the graduate lab of the School of Computer Science at Florida Interna-

Time	TC	Naive	component	msg
1101243861	configuration	configuration	10	Product: Microsoft SQL Server Setup Support Files – Configuration completed successfully.< br >< br >
1101243864	start	configuration	10	Product: Microsoft SQL Server Setup 2005 Beta 2 – Install started.

Table 4: The first example of effectiveness of modified naive Bayes

Time	TC	Naive	component	msg
1101760456	request	request	10	Product: Microsoft Visual J# .NET Redistributable Package 2.0 Beta - - Removal completed successfully.< br >< br >
1101760456	request	configuration	10	Product: Visual Web Developer 2005 Express Edition Beta - English – Removal completed successfully.< br >< br >

Table 5: The second example of effectiveness of modified naive Bayes

tional University. Different gray scales in Figure 1 correspond to different components and different gray scales in Figure 2 represent different categories. In Figure 1, dotted lines, lying in the ellipse range with the same gray scale, are generated from the same component. We observe that these dotted lines in *create* and *connection* categories occur alternately. The dotted line in *dependency* situation overlaps with that in *create* state. From the log file of “raven” we can easily verify that the component “Inventory Scanner” correspond to these lines in the ellipse range. From Figure 1, we observe that the *create* problem of inventory scanner will trigger the *dependency* problem (i.e., it cannot find some features or components.). The continuing occurrences of the *create* and *dependency* problems will finally generate the *connection* problem. Using visualization tools, we note that the *connection* problems occur after the *create* problems for at least three days. However, the *dependency* problems usually appear just several seconds after the *create* problems. The wide range of lags, from seconds to days, also demonstrates the difficulty of using a predefined window for temporal data mining and shows the necessity of methods without a predefined window. In Figure 2, different gray scales represent different categories. The dotted lines lying in the ellipse range of component 6 actually consists of *create* and *dependency* situations which appear alternately. The other lines in component 6 belong to the *connection* category. This scenario depicts the relationships among these three categories. Moreover, from this figure, the dotted line of component 6 is almost in parallel with that of component 5’s, with a small lag. This indicates that component 5 will report problems synchronously whenever there are problems in component 6. In addition, we may need to pay attention to component 7 which seems to cause the problems in component 6 and component 5. The dotted line in component 0 consists of *start* and *report* situations. Using the visualization tool, we observe that after component 0 starts, it tends to keep generating *report* messages. Figure 3 is an example of 3D-plot view of the log data. 3D-plot will help users understand and interpret complicated patterns.

6. CONCLUSION AND FUTURE WORK

In this paper, we present our research efforts on mining log files for computing system management. We propose two approaches for incorporating temporal information to improve the performance of categorizing log messages into common categories. There are several natural avenues for future research. First, instead of manually determining the set of common categories, we could develop techniques to automatically infer them from historical data. For instance, we could use clustering techniques to identify the structures

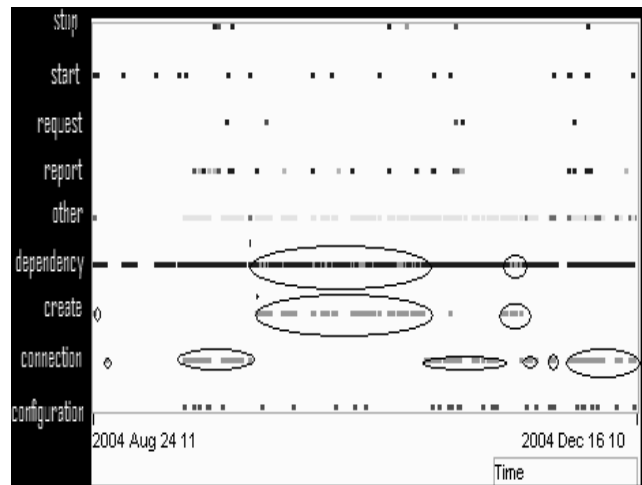


Figure 1: The 2D plot of the “raven” data set. X axis is the time. Y axis is the state

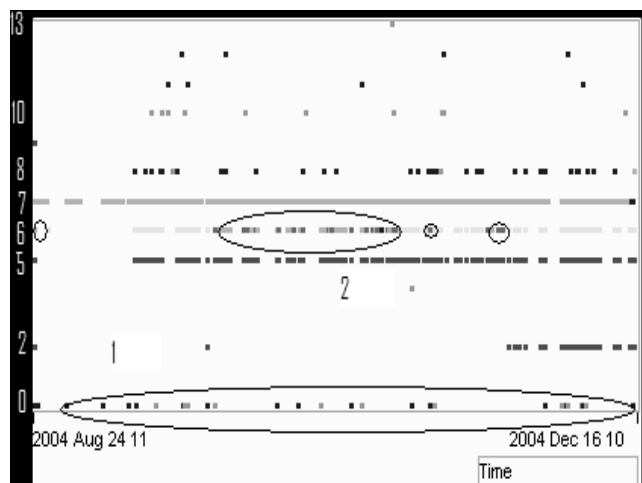


Figure 2: The 2D plot of the “raven” data set. X axis is the time. Y axis is the component

Time	TC	Naive	component	msg
1101237955	report	report	11	Windows saved user N131\Administrator registry while an application or service was still using the registry during log off. The memory used by the user's registry has not been freed. The registry will be unloaded when it is no longer in use. < br >< br >< br > This is often caused by services running as a user account. try configuring the services to run in either the LocalService or NetworkService account.< br >< br >
1101240798	request	configuration	10	Product: Visual Web Developer 2005 Express Edition Beta - English - Installation completed successfully.< br >< br >

Table 6: The third example of effectiveness of modified naive Bayes

Time	configuration	connection	create	dependency	other	report	request	start	stop
1101243864	0.4981456	0.000691	0.001307	0.0000002	0.0036706	0.002910	0.021683	0.4430199	0.0285726
1101760456	0.9495610	0	0.0000046	0	0.0000129	0.0000492	0.0494618	0.0005082	0.0004023
1101240798	0.6447276	0	0.0000031	0	0.0000088	0.000033	0.3526248	0.0006900	0.0019122

Table 7: The text classification probabilities for the above three examples

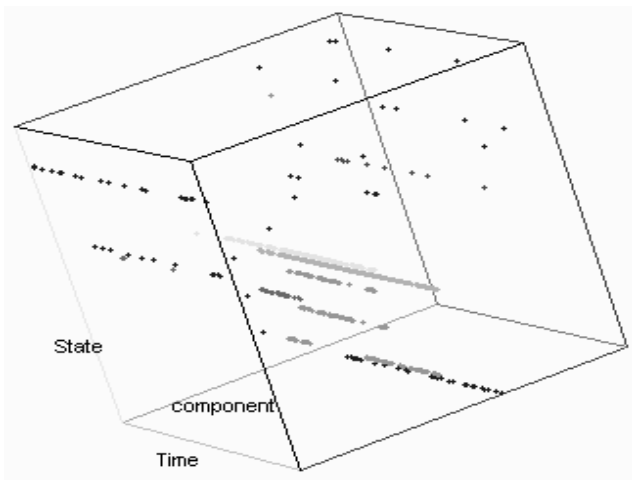


Figure 3: The 3D-plot of the “raven” data set

from large amounts of log data [8]. Second, in practice, the number of different common categories for system management can be extremely large due to the complex nature of the system. To resolve this issue, it would be useful to utilize the dependence relationships among different categories [18; 28]. Third, it would also be interesting to develop methods that can efficiently discover interesting temporal patterns from the transformed log files [1; 14; 16].

7. REFERENCES

- [1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14. IEEE Computer Society Press, 1995.
- [2] Dana Ballard. *An introduction to natural computation*. The MIT Press, 1997.
- [3] M. Chessell. Specification: Common base event, 2003. <http://www-106.ibm.com/developerworks/webservices/library/ws-cbe/>.
- [4] M. Derthick, J.A. Kolojechick, and S.F. Roth. An interactive visualization environment for data exploration. In *Proceedings of Knowledge Discovery in Databases*, pages 2–9. AAAI Press, 1997.
- [5] Dayne Freitag and Andrew McCallum. Information extraction with HMM structures learned by stochastic optimization. *Proceedings of the Eighteenth Conference on Artificial Intelligence*, pages 584–589, 2000.
- [6] J. Goldstein, S.F. Roth, J. Kolojechick, and J. Mattis. A framework for knowledge-based, interactive data exploration. *Journal of visual languages and computing*, 5:339–363, 1994.
- [7] Joseph L. Hellerstein, Sheng Ma, and Chang shing Perng. Discover actionable patterns in event data. *IBM System Journal*, 41(3):475–493, 2002.
- [8] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
- [9] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveiro, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.
- [10] Jeffrey O. Kephart and David M. Chess. The vision of automatic computing. *Computer*, pages 41–50, 2003.
- [11] Nicholas Kushmerick, Edward Johnston, and Stephen McGuinness. Information extraction by text classification. *Proceedings of the IJCAI-01 Workshop on Adaptive Text Extraction and Mining*, 2001.
- [12] T. R. Leek. Information extraction using hidden markov models. Master's thesis, UC San Diego, 1997.
- [13] C. Li and G. Biswas. Temporal pattern generation using hidden Markov model based unsupervised classification. In *In Proc. of IDA-99*, pages 245–256, 1999.

- [14] Tao Li and Sheng Ma. Mining temporal patterns without pre-defined time windows. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 451–454, 2004.
- [15] Sheng Ma and Joseph L. Hellerstein. Eventbrowser: A flexible tool for scalable analysis of event data. In *Proceedings of the 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 285–296. Springer-Verlag, 1999.
- [16] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (SIGKDD'95)*, pages 210–215. AAAI Press, 1995.
- [17] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *In AAAI-98 Workshop on Learning for Text Categorization*, 1998.
- [18] Andrew K. McCallum, Ronald Rosenfeld, Tom M. Mitchell, and Andrew Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 359–367. Morgan Kaufmann Publishers, San Francisco, US, 1998.
- [19] Tom M. Mitchell. *Machine Learning*. The McGraw-Hill Companies, Inc., 1997.
- [20] Wei Peng, Tao Li, and Sheng Ma. Mining logs files for computing system management. In *Proceedings of The 2nd IEEE International Conference on Autonomic Computing (ICAC-05)*. To appear, 2005.
- [21] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77(2):257–286, 1989.
- [22] IBM Market Research. Autonomic computing core technology study, 2003.
- [23] Irina Rish. An empirical study of the naive Bayes classifier. In *Proceedings of IJCAI-01 workshop on Empirical Methods in AI*, pages 41–46, 2001.
- [24] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Survey*, 34(1):1–47, 2002.
- [25] Jon Stearley. Towards informatic analysis of syslogs. In *Proceedings of IEEE International Conference on Cluster Computing*, Sept. 2004.
- [26] Brad Topol, David Ogle, Donna Pierson, Jim Thoensen, John Sweitzer, Marie Chow, Mary Ann Hoffmann, Pamela Durham, Ric Telford, Sulabha Sheth, and Thomas Studwell. Automating problem determination: A first step toward self-healing computing systems. IBM White Paper, October 2003. <http://www-106.ibm.com/developerworks/autonomic/library/ac-summary/ac-prob.html>.
- [27] M.O. Ward. Xmdvtool: Integrating multiple methods for visualizing multivariate data. In *Proceedings of Visualization*, 1994.
- [28] Andreas S. Weigend, Erik D. Wiener, and Jan O. Pedersen. Exploiting hierarchy in text categorization. *Information Retrieval*, 1(3):193–216, 1999.

About the Authors

Wei Peng received her BS degree in Computer Science from Xi'an Institute of Science and Technology, China in 2002. She is currently a doctoral student in the School of Computer Science at Florida International University. Her research interests are data mining and GIS.

Tao Li received his Ph.D. degree in Computer Science from University of Rochester in 2004. He is currently an assistant professor in the School of Computer Science at Florida International University. His primary research interests are data mining, machine learning, bioinformatics, and music information retrieval.

Sheng Ma received BS degree in electrical engineering from Tsinghua University, Beijing China, in 1992. He received MS and Ph.D with honours in electrical engineering from Rensselaer Polytechnic Institute, Troy NY, in 95 and 98, respectively. He joined IBM T.J. Watson research center as a research staff member in 1998. He became a manager of Machine Learning for Systems in 2001. His current research interests are machine learning, data mining, network traffic modeling and control, network and computer system management.