ISSN: 2348 - 2117

International Journal of Engineering Technology and Computer Research (IJETCR)
Available Online at www.ijetcr.org
Index Copernicus Value: 69.88
Volume 6; Issue 5; September-October 2018; Page No. 01-11

System Logs to Learn Error Predictors

N.Rajasekhar Reddy[1], M.Vinay Babu [2]

[1]Department of Computer Science and Engineering,Rayalaseema University, Kurnool, A.P., India

[2]Professor, Department of Coumpeter Science and Engineering, Chalapati Institute of Engineering and Technology, Gntur, A.P,India.

E-mail:- rajasekhar.2802@gmail.com

Abstract

Mitigating the impact of computer failure is possible if accurate failure predictions are provided. Resources, applications, and services can be scheduled around predicted failure and limit the impact. Such strategies are especially important for multi-computer systems, such as compute clusters, that experience a higher rate failure due to the large number of components. However providing accurate predictions with sufficient lead time remains a challenging problem. This paper describes a new spectrum-kernel Support Vector Machine (SVM) approach to predict failure events based on system log files. These files containmessages that represent a change of system state. While a single message in the file may not be sufficient for predicting failure, a sequence or pattern of messages may be. The approach described in this paper will use a sliding window (sub-sequence) of messages to predict the likelihood of failure. The a frequency representation of the message sub-sequences observed are then used as input to the SVM. The SVM then associates the messages to a class of failed or non-failed system. Experimental results using actual system log files from a Linux-based compute cluster indicate the proposed spectrum-kernel SVM approach has promise and can predict hard disk failure with an accuracy of 73% two days in advance.

Keywords: Software maintenance · Data Mining · System Logs · Log analysis · Information Gain · Classification and Prediction of Defective Log Sequences

Introduction

Hardware failures, such as hard disk and processor failures, can impede the execution of applications on High-Performance Computing (HPC) systems since the recovery process can require unexpected amounts of time and resources. The impact of failure is more substantial for large-scale HPC clusters that consist of many computing elements. BlueGene demonstrated that the mean-time to failure of these systems is inversely proportional to the system size (number of computing elements) [1], which results in lower reliability. We can improve reliability by predicting hardware failure and scheduling applications and services around it [10]. This strategy will enable us to fully harness the potential of the next generation HPC. For example Tantawi and Ruschitzka [12] added the concept of an *equicost* checkpointing strategy, which calculates the checkpoint interval by attempting to balance the checkpointing cost against the likelihood of failure. Oliner and Sahoo introduced a risk-based model that incorporates failure indicators from real job logs, communication topologies, scheduling policies, and real failures to predict processor failure. They found that risk-based cooperative checkpointing with prediction accuracies of only 10% can still produce performance improvements [8]. However, the success of this strategy will ultimately rely on accurate predictions of imminent failures with sufficient lead time to efficiently mitigate failure. There have been several approaches for predicting system failure using system log files [4, 8, 11, 15, 14]. System log files consist of messages created by the different processes executing on the system. The information recorded varies from general messages concerning user logins to more critical warnings about program failures. Prediction methods include standard machine learning techniques such as Bayes networks, Hidden Markov Models (HMM), and Partially Observable Markov Decision Process (POMDP) [14]. The use of time-series analysis is common among these methods since a system message in isolation has been shown to be

Corresponding author: N.Rajasekhar Reddy

insufficient for predicting failure [9, 15]. We also believe that certain sequences of log messages may provide sufficient information to predict failure.But the large amount of information available in system log files makes finding the right pattern(s) difficult. In this paper we introduce a new approach for predicting critical system events based on Support Vector Machines (SVMs). Given labeled training data, an SVM

Keywords Software maintenance · Data Mining · System Logs · Log analysis · Information Gain · Classification and Prediction of Defective Log Sequences dict system failures (Pinheiro et al. 2007; Jiang et al. 2009a), whereas a set of events preceding it may do (Featherstun and Fulp 2010). These sets of events merit the attention of system managers either because immediate action must be taken or because there is an indication of an underlying problem (Oliner and Stearley 2007). Log analysis is a science that aims at interpreting logs. One of the applications of log analysis is to feature sequences of events that can be associated with system / computer failures (Oliner et al. 2012; Forrest et al. 1996; Pinheiro et al. 2007; Yamanishi and Maruyama 2005; Oliner and Stearley 2007; Li et al. 2007). A sequence is a set of events ordered by their time-stamp and occurring within a given time window. In particular, log analysis classifies sequences by their number of errors and use this information to make prediction on future sequences. A solution of such classification problem is a predictor, i.e., a machine that makes predictions, and a characterization of the sequences used by the machine. The characterization can be employed to isolate undesirable behaviors of the system. Accuracy of prediction is also known and can be used to compare different solutions on novel data provided settings and data are similar to the past as we illustrate in Section 11.2. In these settings, log analysis has some known challenges: 1) log data can be huge and manual inspection can be unrealistic (Oliner et al. 2012; Nagappan et al. 2010), 2) logs can be coded in a cryptic language that does not help their interpretation (Oliner and Stearley 2007; Oliner et al. 2012), 3) the structure of log sequences and its length can be hard to define (Oliner et al. 2012; Featherstun and Fulp 2010). In this study, we target some of these challenges. Contribution to research progress in log analysis. For this study, we learn error predictors from sequence abstractions, i.e., representations of sequences in some formal format that machines can read. The method we propose builds predictors that managers and practitioners can easily use to perform risk analyses, as suggested by Hall et al. (2012). Our approach uses three wellknown Support Vector Machines (SVMs) as predictors and proposes: – A sequence abstraction based on information carried by logs – An analysis walk path that integrates different statistical techniques to control for data brittleness and accuracy of model selection and validation. – A parametric classification problem that varies according to different degree of defectiveness. A sequence abstraction is a representation of log sequences according to given rules. In this study, we propose to read logs by application and represent log sequences by type and quantity of information carried by its events. Context and system operational profile de- fine rules to set the amount of such information. Unlike recent studies (Fronza et al. 2011; Shang et al. 2013), sequences with the same abstraction are not excluded in our analysis. On the contrary, sequence abstractions are weighted by the times they occur and the number of users whose interaction with the system triggered the events. Weights give to sequences abstractions different relevance in the classification problem. In this study, sequence type and its weights are the input features for the SVMs. RQ1: Is the amount and type of information carried by a sequence enough to predict errors? We aim at demonstrating that the abstraction we propose suffices to obtain predictions superior to previous studies. Other abstractions may additionally consider time ordering among the events (Fronza et al. 2013), use sliding time windows (Vilalta and Ma 2002), also in combination with tag numbers (Featherstun and Fulp 2010; Fulp et al. 2008), or filter sequences by thread. Such abstractions are more complex to build and still miss relevant cases like failures due to the interaction of different threads, e.g., parallel access to the same resource. Our analysis introduces to the use of different techniques that researchers must consider ensuring the scientific rigor of their solution. After data cleansing, our approach provides a formal definition of input and output of machine learners, reduces the input feature spaces by selecting input variables relevant to failures leakage, manipulates the original sequence sets to create samples for model cross-validation, and applies the three SVMs to full

and reduced feature spaces. Each technique is discussed against its typical challenges (Section 4). In some cases, we compare different techniques solving the same problem. In particular, we investigate the following questions: RQ2: Does feature reduction increase the quality of prediction? Specifically, we discuss whether selecting input variables with Information Gain (Quinlan 1987) improve true positive rate, false positive rate, and precision of the SVMs on novel data. Information Gain selects features that contribute to the information of a given classification category. In our case, such category consists of defective sequences. According to Zhang and Zhang (2007), if classification categories are not equally represented in data sets, classifiers might have low precision even though true positive rate is high and false positive rate is low. Such imbalanced data sets are very frequent in software engineering data (Menzies et al. 2007b). Khoshgoftaar et al. (1997) have introduced a technique of set balancing to improve the quality of prediction. We discuss this technique with our data sets. With our method, we recommend to select models that accurately fit historical data before using them for predictions. In principle, being not accurate in fitting historical data gives researchers a larger set of models on which the chance to find a model with higher prediction accuracy increases. It might also lead to the undesirable consequence of model under-fitting or over- fitting, i.e. to use models with a complexity that does not correspond to the one of the actual system behavior (Hall et al. 2012; Alpaydin 2010). Literature on mining system logs typically neglects to report fitting accuracy of predictors preventing the reader to really understand the value of the research contribution. In error prediction, classifying log sequences is a parametric problem. The parameter defines the number of errors a sequence must contain in order to be classified as defective. We call it cut-off. The cut-off depends on two factors: research problem and data distribution. In some environments, only sequences with two errors or more deserve some attention, (Oliner and Stearley 2007). In general, the definition of defective sequence depends on the degree of system reliability we want to analyze. Varying the cut-off value changes the sequence distribution over the classification groups, which in turn influences convergence and precision of predictors (Khoshgoftaar et al. 1997; Zhang and Zhang 2007). Studying prediction at different cut-off values helps understanding convergence and prediction ability of classifiers: classifiers that perform the best for all given cut-off values, solve the prediction problem independently from the level of reliability requested for the system. To our knowledge, there is no literature in log analysis with cut-off value two or more. On the opposite, literature on fault prediction is reach of studies that classify software modules by two or more faults. Table 1 reports some of them. Overall, the major contributions of this work to the classification problem are: – Sequence abstraction. As we mentioned, we propose to characterize sequences by the information they carry - type of event, occurrences of events, occurrences of sequences, and number of users - eventually reducing it with statistical techniques. The result is promising and tells that more sophisticated approaches for sequence abstraction, like considering time ordering, might not be necessary. – Parametric definition of the classification groups. We can identify classifiers that are good predictor at different error degrees. We can also explore the classification problem with different balance in the output categories as changing the cut-off value can potentially change the distribution of sequences over them. – Parametric splitting of the original data sets for cross-validation. Parametric splitting contributes to control the influence of splitting size on classification performance and increments the number of sets on which models can be cross-validated and evaluated. As implication, a model that consistently performs the best over set triplets obtained by different parameter values is a more reliable predictor for the original data sets. – Balance splitting of the original data sets to represent equally output categories. In this case, the best result we get is to increase convergence of classifiers. The technique is not applicable to all the data sets, though. Application of the method and context-specific questions. To discuss all these issues and illustrate the application of our method, we analyzed a real system for telemetry and monitoring of race cars. The specific system is a suitable test bed for our method as it has stringent requests of reliability.

The company owning the system was interested in a predictor that could be easily used. Thereafter, we selected three well-known SVMs and asked for our case study: BQ1. Can we use Support Vector Machines to build suitable predictors? For the same reason, we were also interested in understanding whether a single type of learning machine can be used for all applications of the system and for different cut-off values: BQ2. Is there any Support Vector Machine that performs best for all system applications? Is there any machine that does it for different cut-off values? These Business Questions address the specific requests of the company, but can also contribute to understand abilities and limitations of our research work. In the end, in our case study, we found classifiers that: – At individual application, have prediction ability superior to existing literature on log analysis reaching performance of 1% false positive rate, 94% true positive rate, and 95% precision, Table 15
and 84% precision, Table 14. As we will see, this result is overall satisfactory despite the moderate low value of the true positive rate. Namely, the value is attained with high precision on twenty-five data sets with highly varying distributions over the two classification categories. – When we restrict to models that predict with balance greater than the average, we are able to find ten data sets for which best models have average performance of 9% false positive rate, 78% true positive rate, 95% precision, Table 17. Overall, the major results that this study provided to managers of the company are: – Identify the type of events that mainly contributed to the occurrence of errors in a sequence. We did it for any number of errors the managers set for a sequence to be defective (e.g., ”sequences of logs must have at least three errors to raise any alarm”). – Provide predictions on types of costs of maintenance of the telemetry system or some of its applications. We did it with the degree of confidence that eventually managers request (e.g., “provide predictions that
Table

Table 1: Some literature on data mining for classification of defective modules

Study	cut-off	defective modules
Khoshgoftaar et al. (1997)	3	14 %
Liu et al. (2010)	1	5-20 %
Denaro and Pezzè (2002)	4	about 23%
Porter and Selby (1990)	N/A	25%
Xing et al. (2005)	10	40%
Le Gall et al. (1990)	N/A	5-20 %
Nagappan et al. (2010)	1	N/A
Menzies et al. (2007b)	1	0.4-49%

Per this strategy, there is an a priori classification of modules into two mutually exclusive groups. A criterion variable is used for the group assignment. For example, a module is classified with a code of zero if it has been found defective, or with a code of 1 otherwise. A classifier computes the posterior probabilities of group membership. For example, Munson and Khoshgoftaar (1992), and Nagappan et al. (2008), use the logistic probability of module attributes. The module is then assigned to the group for which it has the greatest probability of membership. False and true positive rates, precision and true positive rate are then calculated to evaluate the performance of the posterior probabilities against the a priori classification. In this notation, what we propose is a priori classification whose criterion variable depends on a cut-off value. Thus, for example, a sequence is classified with a code of zero if it has less than a cut-off value, and with a code of 1 otherwise. Table 1 lists cut-off values in benchmark literature on classification of software modules. For log sequences, to the best of our knowledge, there is no research with cut-off greater than one (Fulp et al. 2008; Featherstun and Fulp 2010). Increasing the cut-off value allows to study classification problems in which the distribution of groups is highly imbalanced, as in the original paper of Munson and Khoshgoftaar (1992). Imbalanced data can be also an issue when we want to assess the performance of posterior probabilities. According to Zhang and Zhang (2007), the measures introduced in Menzies et al. (2007b), (true and false positive rate and balance) are not sufficient in case of imbalanced data. When data is imbalanced, Zhang and Zhang prove models can be able to detect faults (true positive rate (TPr) is high) and raise few false alarms (false positive rate (FPr) is small) but still be very poor

in performance (precision is small). According to Menzies et al. (2007a), precision is an unstable measure and should not be used alone. The authors show that precision computed for different types of learning machines and applied on the same data sets shows the highest variation among the performance measures and using it is more risky. The authors also illustrate how classifi- cation results with low precision and high true positive rate are useful and very frequent in software engineering. Table 1 shows further studies that we found to support this claim. Few recent studies have classified log sequences to predict system failures. Table 2 illustrates the major characteristics of these studies according to sequence abstraction, models used, set up, and performance measures. Among these studies we selected the ones that use SVM to serve as baseline for our work: Liang et al. (2007); Fulp et al. (2008); Fronza et al. (2011, 2013). As all these papers do not report the goodness of the fitted model, we can only use them to compare our result on prediction performance. The major difference among the studies is the way sequences are abstracted and features to feed the SVMs are defined. None of the studies pay specific attention to cross-validation, imbalance distribution over the output groups, or consider feature reduction. None of them include number of users or duplicates in the definition of sequence abstraction, which, according to Oliner and Stearley (2007), are useful to

Table 2: Comparison Studies on system logs

Study	Sequence abstraction	Model(s)	Set up	Measures of prediction performance
Fronza et al. (2011)	Context rules and vectors of event multiplicities. Duplicates are discarded	SVM with Linear and Radial Basis Function vs. Cox Proportional Hazard Model (Cox 1972) as in Li et al. (2007)	Six real data sets from one system. Sequences are sampled with Monte Carlo method to obtain 60-40% cross-validation	Cox model outperform SVMs, but still perform unsatisfactory in some data sets. Best result: Cox model $TP_r = 97\%$, $FP_r = 25\%$
Fronza et al. (2013)	Context rules and Random Indexing Sahlgren (2005) to include time ordering. Duplicates are discarded	Regular vs. Weighted (i.e., uneven cost matrix) SVMs with Linear, Polynomial, and Radial Basis Function kernels	Six real data sets, the same sets of Fronza et al. (2011). Sequences are sampled with Monte Carlo method to obtain 60-40% cross-validation	Weighted SVMs outperform regular SVMs. No unique classifier type has been found across all six data sets. Best result RBF Weighted $TP_r = 93\%$, $FP_r = 2\%$
Vilalta and Ma (2002)	Sequence patterns as matching events in time windows preceding a target event	Association rules and sliding time windows	Synthetic and real data. For each sequence 50% of events serve for training and 50% for testing. Replicated on 30 different sequences	MR and FN_r. MR reduces with the number of events in a pattern. FN_r significantly decreases as time window size increases. The way it drops depends on the target event
Salfner et al. (2006)	Short time series of error messages are clustered by similarity and used to train and predict delays in response time	Three time models: Semi-Markov chain with enriched state characterization, reliability growth Poisson model, Dispersion Frame heuristic technique	One real data set. Time series analysis on 30 sec. time windows	Precision= 80%, TP_r = 92.3%, and F-Measure= 85%.
Li et al. (2007)	Frequent failure signatures: sequence of ordered or partially ordered event sequence segments in sliding time window. Defined similarly as in Vilalta and Ma (2002)	Cox Proportional Hazard Model, Cox (1972)	One synthetic and one real data set	Akaike Information Criterion. The model predicts effectively on long event sequences
Liang et al. (2007)	Sequence of events defined by severity levels and text message and defined in a sliding time-window	RIPPER (a rule-based classifier), SVM with the Radial Basis Function kernel, and two Nearest Neighbor models	Real data set from IBM BlueGene/L super computer. Data collected during 22-weeks split into training weeks (19 weeks) and testing weeks (three weeks). 15 tests varying the testing weeks.	The customized Nearest Neighbor model outperforms the other classifiers. Predictive ability depends on time window. The longer the window the better is the prediction for all the classifiers. Customized Nearest Neighbor method: F-Measure $\geq$ 60%, Precision $\geq$ 50%, and $TP_r \geq 70\%$

The authors obtain a very good performance of the Cox model on only one of their six data sets. This might be due to several contextual / construction reasons as, for example, to some kind of ”brittleness” of data, i.e. changes in data used to learn predictors (Menzies et al. 2007b), or inaccuracy of data pre-processing (Gray et al. 2011). Lo et al. (2009) compare two different sequence abstractions in the classification of execution traces of software programs: one derived from the information of a sequence (as in our sequence type, but not considering sequence occurrences or users) and the other derived from iterative patterns defined as sub-sequences with the same initial and final points. The authors prove that feeding learning machines with the latter abstraction increases the performance of the models by 24,68%. They use synthetic and real data on which they run test cases for specific bugs they injected. They run their analysis with LIBSVM library (Chang and Lin 2011) and a non-specified type of kernel. Their new abstraction is particularly suitable for execution traces where the level of details in logs allows, for example, identify closed substructures (e.g., loops) attributable to specific behaviors (e.g., failures) or define the length of a sequence by the structure of test cases. System logs, instead, have a higher level of detail that cannot be used to identify small substructures. Encoding execution logs into features readable by SVMs needs sophisticated techniques (e.g. text mining) as log messages are often in free-format text. To standardize them, Jiang et al. (2008, 2009b) and Shang et al. (2013) divided static from dynamic information in logs and aggregate sequences by their dynamic information. The resulting sequence abstraction neglects any form and information from duplicates: for example, two similar sequences with just different users are counted as one. As we mentioned, we instead weight sequences by the number of users they have. In terms of dynamic and static information, our abstraction has memory of the dynamic information. Given the different level of details in system and execution logs, our form of abstraction does not require as much effort as for execution logs. For example, in our case study, task descriptions are standardized into one / two words, Table 1. This reduces by far concerns on log encoding. 3 Context Our analysis has been performed with industrial data of a large company producing cars. The company prefers to be anonymous and for the rest of the paper, it will be referred to as SoftCar. SoftCar has an internal unit dedicated to the development and maintenance of software for production, test, and telemetry of cars' performance. Managers of SoftCar want to understand system and applications' performance to characterize sequences of events that lead to errors and predict cost of inspection. SoftCar uses the Microsoft Team Foundation Server (TFS), a client server software to manage and track the activities of its 92 applications used in the development and maintenance of software. In particular, TFS uses Windows Event Viewer as monitoring tool. 3.1 Logs From TFS, we collected logs stored in seven servers and sent by 974 PCs used by 876 different users in threemonths (December 2008-February 2009). Overall, we were able to collect 3,01 GB of data for 50 different applications used in the three months period. Data has been stored in a MySQL database and mined with the R tool1 . Each event carries information on its arrival time (Date), the application that triggered it (Application), the machine from which it was triggered (Computer ID), the server that has stored it (Server ID), the user that logs it (User ID), a project identifier (Area name), its task (Event type), and its state (State). In addition, some of the log entries (not all) include also an event message (Event description). Events are stored in servers and sent by applications from different computers and different users. The Area name is a typical built-in description in TFS that describes the project competence and will not be used in this work. Table 3 illustrates a sample of an original system log file. With filtering system logs by application name, we obtain a data set per Application. Each data set consists of sequences of events performed by some loggedin users from some computers and recordedinto one or more servers of the company. Sequences are defined as sequential set of events. Defective sequences are sequences containing at least one event with state “Error”. Defective sequences can have more than one event in error state. The example in Table 3 shows two full sequences: the former starting at 2009-03-02 07:05:45 and ending at 2009-03-02 07:05:46 and the latter starting at 2009-03-02 07:06:45 and

ending at 2009-03-02 07:14:58. The former is composed by two events of different types stored in the same server and sent from one PC and user. The latter is a sequence with two errors composed by four events of three different types stored in the same server, sent by two different

8

Table 3: Sample application logs of the telemetry system

Date	Server ID	PC ID	User ID	State	Type	Event Description
2009-03-02 07:05:45	1472	36248	26209	Information	Log In	Application LogOn
2009-03-02 07:05:46	1472	36248	26209	Timer	Systems	Application Connection Init.
2009-03-02 07:06:45	1472	26210	1863	Information	Log In	Time Stamp
2009-03-02 07:06:45	1472	26210	1863	Information	General	Generic Information
2009-03-02 07:10:20	1472	5776	19039	Error	General	Generic Error
2009-03-02 07:14:58	1472	5776	19039	Error	Performance	Generic Error

4 Method Outline Our goal is to select classifiers that predict defective sequence types with the best accuracy possible. The method is an enhancement of the analysis pattern presented in Russo (2013). It combines data pre-processing, feature reduction, parametric sample splitting, classi- fication, and cross-validation. The method is applied to every application A and with three types of classi- fiers: a neural network multilayer perceptron with backpropagation algorithm and sigmoid activation function (MP), a neural network radial basis function (RBF), and a linear network (L). In the following, we introduce the different stages of our method. Each stage is further detailed and implemented in our case study in the corresponding Sections 5-9. 4.1 Data Pre-processing, Section 5. A classifier is a model that splits input into predefined output categories (or groups). The set of all inputs is called feature space. The output consists of categories that group input according to a certain membership criterion. As we mentioned, the action to associate each feature with its true category is called a priori classification. The major goal of data pre-processing is to determine the feature space and the output categories. In this case study, there are as many feature spaces as the number of applications used in the system: 50 feature spaces of which 25 have representatives in both categories. As there is little overlap of event types across applications (see Section 5.1), we decided to replicate our analysis on each application independently. We define the two output categories in terms of the cut-off parameter, c. Any new value of c determines a new classification problem. For example, if c = 3, we classify features into the ones that have more and those that have strictly less than three events in error state. 4.2 Feature Reduction, Section 6. Feature spaces can be reduced to decrease the complexity of classifiers (known as overdimensionality of the feature space). In their systematic literature review on fault prediction performance in software engineering, Hall et al. (2012) report that feature reduction improves performance of models. Typically, features can be reduced in two ways: with wrappers that use classifiers for heuristic search in the space of all possible feature subsets or with filters that apply statistical analysis to reduce the number of features selected and then build the classifier with them. Wechose the latter approach and filtered the feature spaces using Kullback Leibler Information Gain (IG) (Quinlan 1987; Menzies et al. 2007b). In information theory, IG is typically used to define a measure of correlation, which also generalizes the usual productmoment correlation coefficient (Kent 1983). IG measures the bits required to encode a class after observing the effect of a variable. Variables are ranked according to the gain from the most informative and SVMs learn from subsets defined by the top ranked variables. IG is fast and simple. Unlike other techniques like standard principal component analysis, IG also captures nonlinear dependencies among variables. Variables that do not contribute to information are eliminated. In our study, we further compare the performance of SVMs on full and reduced feature spaces to discuss whether feature reduction with IG can increase classi- fier performance (RQ2). With IG, we may also know which are the most informative features and identify sequence characteristics (e.g., event types) that contribute to the prediction of errors in logs the most. 4.3 Parametric Sample

Splitting, Section 7 To perform any classification, we need to control curse of dimensionality (Aliferis et al. 2010). This phenomenon occurs since the data points required for modeling grow exponentially with the number of variables (Bishop 1996; Alpaydin 2010). Hence, models obtained from limited data may perform poorly. Techniques of sample manipulation often in combination with feature reduction (Section 6) limit this effect and identify the minimumsize subset of variables that exhibit the maximal predictive performance (Guyon and Elisseeff 2003). To control for curse of dimensionality in the case of MP, Khoshgoftaar et al. (1997) run the classifier on a triplet of sets (training / test / validation sets) obtained from the original set with a particular splitting technique. The three sets provide fresh data for training, fitting and generalizing models.

Table 4: Information used to build sequence abstrac tions from Table 3

Seq.	Ev.	Date	User	State	Type
s_1	e_1	2009-03-02 07:05:45	26209	Information	Log In
	e_2	2009-03-02 07:05:46	26209	Timer	Systems
s_2	f_1	2009-03-02 07:06:45	1863	Information	Log In
	f_2	2009-03-02 07:06:45	1863	Information	General
	f_3	2009-03-02 07:10:20	19039	Error	General
	f_4	2009-03-02 07:14:58	19039	Error	Perform

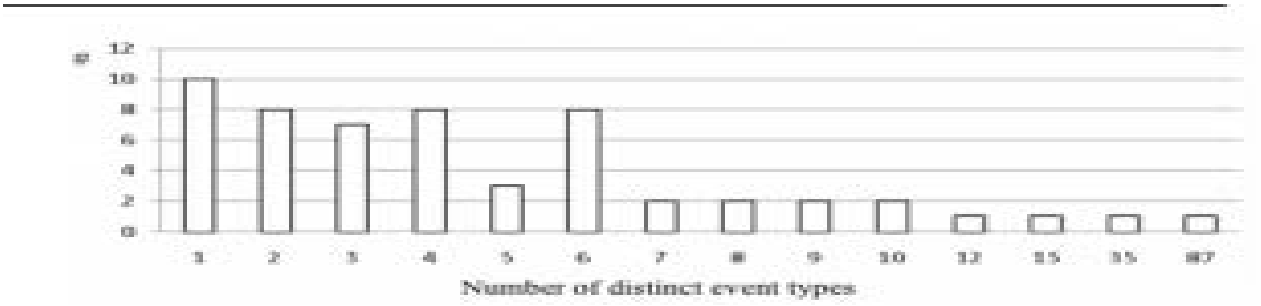

Fig. 1: Number of applications (y-axis) with a given number of event types in our system

A sequence starts with an event after one or more consecutive events in “Error” state. – A sequence ends with one or more consecutive events in “Error” state, – A sequence ends immediately before an event of type “Log-in”, or – A sequence ends with the end of the day. Each sequence is a set of events eventually repeated. For example, in Table 4, there are two sequences, s1 and s2. s1 has two events of type “Log-in” and “Systems.” s2 has four events of three types {General, Login, Performance} and ends with two events in “Error” state. To define sequences and their length, other approaches use sliding time-windows (Liang et al. 2007; Vilalta and Ma 2002; Li et al. 2007; Fulp et al. 2008). In this case, the analysis must be repeated on time-windows of different size. Using context rules prevents analysis repetitions, but requires interaction with system users and administrators to extract and validate the rules. We could do it, but in other systems, this might not be feasible. 5.1.1 Sequence Type In this section, we define all the major terms that will be used in the analysis. Table 4 and 5 illustrate an example that we will use to guide the reader in the construction of a sequence type and its variables that together constitute our sequence abstraction. Formally, we define the multiplicity μ of an event type, e, as the number of times it occurs in a sequence s. Under this

definition, each sequence s is mapped into a vector of multiplicities of its event types For each value of c and each set ST, we computed the percentage of c-defective sequence types. Fig. 3 shows that, for c = 1, ..., 4, the percentage depends on the size of ST in a super linear fashion. According to Le Gall et al. (1990) and Khoshgoftaar et al. (1997), classifiers better converge if data in each category ranges by between 5 and 20% and they perform even better when data is balanced in the two groups (i.e., 40-60% range, (Xing et al. 2005)). Examining Fig. 3 and Table 6, we have – For c = 1, the percentage of defective sequence types is above 20% in the majority of the 25 sets – For c = 1, all (eight) small sets and for c > 1, about half of large sets fall in the 5-20% range – Sets that are balanced are rare and they 1

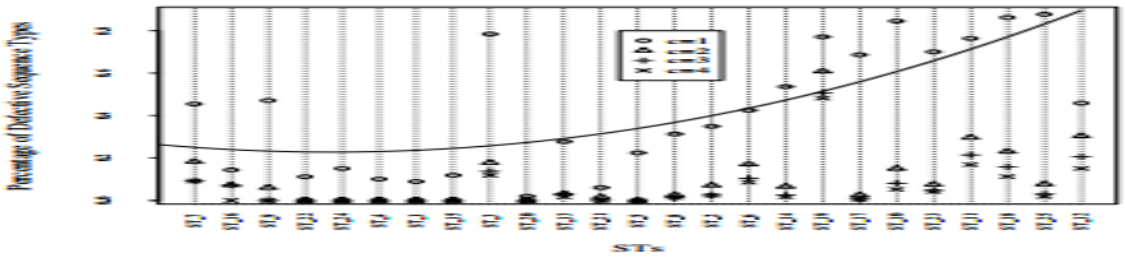

Fig. 3: Percentages of c-defective sequence types over applications. c=1,...,4. ST_i are ordered by their size.

Table 7: Number of attributes, original and remaining after applying IG. Seven data sets are missing, as they have not been reduced. * or ** indicate the presence of either μ or ν and μ, respectively. "-" indicates no reduction. Avg. Red indicates the average reduction percentage over cut-off values.

ID	Application Type	Orig. Attr. #	Red. Attr. #				Avg. Red.
			c=1	c=2	c=3	c=4	
ST_1	Telemetry Module	5	2	-	-	-	15%
ST_2	Telemetry Module	9	3	4*	1*	1*	75%
ST_3	Telemetry Module	7	1	2*	2*	3*	71%
ST_5	Sw. Resources Mgmt.	8	2	-	-	-	19%
ST_6	Product Sw. Tools Mgmt.	12	7**	6**	6**	5**	50%
ST_7	Procurement Sys. Module	4	1	-	-	-	19%
ST_{10}	Telemetry Module	12	-	5**	5**	2*	50%
ST_{11}	Product Data Mgmt.	6	4*	4*	4*	4*	33%
ST_{12}	Chain Supply Mgmt. Sys.	89	69**	73**	74**	74**	19%
ST_{13}	Procurement Sys. Module	6	1	-	-	-	21%
ST_{14}	Procurement Sys. Module	10	4*	5**	1*	1*	73%
ST_{15}	Data Transfer Module	8	-	1	-	-	22%
ST_{17}	Product Sensors Mgmt.	11	-	2*	-	-	20%
ST_{18}	Telemetry Module	11	4*	6**	6**	6**	50%
ST_{19}	Secondary DB	5	4*	4*	4*	4*	20%
ST_{21}	Virtual Disk Service Module	10	1	-	-	-	23%
ST_{23}	Manufacturing Execution Sys.	17	11*	14**	10**	10**	34%
ST_{25}	Virtual Disk Service	37	19*	19**	16**	15**	52%

Future Work

Our future work will develop around two major points: use of other promising classifiers and replication of the study in other environments. – Use of other promising classifiers. Other models have been proven to be good predictor of faults in software modules, like the Bayesian models (Hall et al. 2012). The ability of these models consists in being more robust to the variation of data (Menzies et al. 2007b). – Replication of the study in other environments. As any empirical study on a single context, the results of the application of our method are limited to the specific environment of the study. As we mentioned, a definition of sequence abstraction over system versions or across different systems is still an open research field and is important in the automation of log analysis. Future work will use data sets from different software projects as well as other domains Conclusions

We proposed a method to mine system logs and learn error predictors and applied it to a real telemetry system. We defined log sequences and their abstraction according to the information they carry. We created feature spaces of sequence abstractions that we used with three well-known SVMs. We input the SVMs with full and filtered feature spaces to observe any change in the accuracy of predictions. We introduced parametric splitting to control for data size, curse of dimensionality, and the distribution of features over the classification groups. We observed that 1) a simple abstraction based on the information carried by event logs is enough to obtain satisfactory predictions even across the applications of a system, 2) feature reduction helps identify best performing classifiers independently from data manipulation used in cross-validation, confirming the results in Menzies et al. (2007b) and Hall et al. (2012), and is crucial to determine key input variables for the classification problem (Khoshgoftaar et al. 2010; Menzies et al. 2007b; Guyon and Elisseeff 2003), 3) data splitting is beneficial to classification problems as "feature selection based on sampled data resulted in significantly better performance than feature selection based on original data" (Khoshgoftaar et al. 2010), and 4) parametric data splitting increments the diversity of the sets and the resulting robustness of the classification (Menzies et al. 2007b). In the specific case of the telemetry system, we observed that the classification problem of log sequences is non-linear. Linear regression poorly performs over data sets, and, in fact, best performs in case of feature spaces with few c-defective instances for which the separation problem is more straightforward and can be accomplished by a hyper-plane. In our study, the classification analysis also shows that when performance is set at high values for quality of fit, the multilayer perceptron outperforms the radial basis function and linear classifier. In this case, the radial basis function classifier is more resilient on data sets with varying percentages of c-defective instances and the multilayer perceptron performs the best on data sets with balanced instance representatives or a large number of c-defective instances as discussed in Jayawardena et al. (1997). On average across the applications, we observe the predominance of the multilayer perceptron classifier both for quality if fit and prediction accuracy. We see that classifiers give the best prediction for sequences that have more than three errors. This result might be used to understand and forecast the overall reliability of the system. Our analysis also shows that 1) misclassification rate alone is not a good measure to compare the overall performance and the balance operational measure can be used instead, 2) for comparison over different applications, the use of three measures, true and false positive rates and precision helps compare results on different data sets, as suggested in Zhang and Zhang (2007); Menzies et al. (2007a).

References

1. C. F. Aliferis, A. Statnikov, I. Tsamardinos, S. Mani, and X. D. Koutsoukos. Local causal and markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation.
2. J. Mach. Learn. Res., 11:171–234, March 2010. ISSN 1532- 4435. E. Alpaydin. Introduction to Machine Learning. The MIT Press, 2nd edition, 2010. C. M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, USA, 1 edition, January 1996.
3. C. C. Chang and C. J. Lin. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm. D. R. Cox. Regression models and life-tables. Journal Roy. Statist. Soc. Ser. B. Methodological, 34:187220, 1972.
4. G. Denaro and M. Pezz`e. An empirical evaluation of faultproneness models. In Proceedings of the 24th International Conference on Software Engineering, ICSE '02, pages 241– 251, New York, NY, USA, 2002. ACM. R. W. Featherstun and E. W. Fulp. Using syslog message sequences for predicting disk failures. In Proceedings of the 24th international conference on Large installation system administration, LISA'10, pages 1–10, Berkeley, CA, USA, 2010. USENIX Association.
5. R. A. Finan, A. T. Sapeluk, and R. I. Damper. Comparison of multilayer and radial basis function neural networks for text-dependent speaker recognition. In Neural Networks, 1996., IEEE International Conference on, volume 4, pages 1992 –1997 vol.4, jun 1996. S. Forrest,
6. S. A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for unix processes. In Security and

Privacy, 1996. Proceedings., 1996 IEEE Symposium on, pages 120– 128, 1996. I

7. Fronza, A. Sillitti, G. Succi, and J. Vlasenko. Failure prediction based on log files using the cox proportional hazard model. In Proceedings of the 23rd International Conference on Software Engineering & Knowledge Engineering (SEKE'2011), Eden Roc Renaissance, Miami Beach, USA, July 7-9, 2011, pages 456–461, 2011.