Noname manuscript No.
(will be inserted by the editor)

Mining System Logs to Learn Error Predictors: A Case Study of a Telemetry System

Barbara Russo · Giancarlo Succi · Witold Pedrycz

the date of receipt and acceptance should be inserted later

Abstract Predicting system failures can be of great benefit to managers that get a better command over system performance. Data that systems generate in the form of logs is a valuable source of information to predict system reliability. As such, there is an increasing demand of tools to mine logs and provide accurate predictions. However, interpreting information in logs poses some challenges.
This study discusses how to effectively mining sequences of logs and provide correct predictions. The approach integrates different machine learning techniques to control for data brittleness, provide accuracy of model selection and validation, and increase robustness of classification results. We apply the proposed approach to log sequences of 25 different applications of a software system for telemetry and performance of cars. On this system, we discuss the ability of three well-known support vector machines - multilayer perceptron, radial basis function and linear kernels - to fit and predict defective log sequences.
Our results show that a good analysis strategy provides stable, accurate predictions. Such strategy must at least require high fitting ability of models used for prediction. We demonstrate that such models give excellent predictions both on individual applications - e.g., 1% false positive rate, 94% true positive rate, and 95% precision - and across system applications - on average, 9% false positive rate, 78% true positive rate, and 95% precision. We also show that these results are similarly achieved for different degree of sequence defectiveness. To show how good are our results, we compare them with recent studies in system log analysis. We finally provide some recommendations that we draw reflecting on our study.

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

1 Introduction

System failures can cause unscheduled system downtime and unplanned maintenance costs. When system administrators can predict failures, they can better allocate resources and services to limit these costs. Data that systems generate, typically in the form of logs, can be used to gain a better insight into system behavior and make failure predictions. Mining logs is not an easy task, though, as information in logs does not have a standard structure. There is an increasing demand of better tools to mine logs that provide reliable failure predictions (Oliner et al. 2008).

Logs track changes of system states, so-called events, through monitoring tools and their logging service, e.g., Windows event logs or Linux Syslog (Featherstun and Fulp 2010). Typically, a log event carries information on the application that generated the event and its state, the task and the user whose interaction with the system triggered the event, and the time-stamp at which the event is generated. As such, logs contain information on system behavior generated by its users. Some behaviors are desirable and some are not. In literature, undesirable behaviors are referred as system failures (Lo et al. 2009). System failures can be crashes that immediately stop the system and are easily identifiable as well as deviations from the expected output that let the system run and reveal only at completion of the system tasks.

The manifestations of failures in logs are events in error state that we call errors. Errors act as alerts. Alerts can have originated from a series of events preceding the error; so a single event may not suffice to pre-

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 well-known 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 define 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.

RQ3: Does set balancing increase the quality of prediction?

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.
- Across applications, on average, have prediction ability of 4% false positive rate, 41% true positive rate,

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 render the current behavior of at least of 70%") and a high degree of precision. In particular, we were able to forecast the percentage of higher costs of inspection, i.e. costs for developers to inspect sequences erroneously predicted as defective or costs for non-budgeted maintenance activities, i.e. costs for fixing errors that were not predicted.
– Characterize the above costs for different levels of defectiveness set by the managers. We found that costs for non-budgeted maintenance activity were limited for sequence with one error but increasingly higher when sequences have more errors. We also found that costs of inspection for sequences erroneously predicted as defective are medium low when they have one error and decreasingly lower when they have more errors, which suggests to focus inspection on sequences with more than one error.

In Section 2, we discuss research relevant to this study. In Section 3, we present the context of study. We outline the method in Section 4 and present data and preprocessing techniques in Section 5. We introduce feature reduction and parametric sample splitting in Sections 6 and 7, respectively. Finally, we introduce classifiers and their measure of performance in Section 8. We present our strategy for cross-validation in Section 9. We summarize our findings in Section 10 and answer to research / business questions in Section 11. In this section, we further compare our approach against existing literature on prediction in log analysis. In Section 12 and 13, we discuss the threats to validity and the future of our study. Conclusions are finally drawn in Section 14.

2 Related Work

Logs are one of the major data sources used in building prediction models in empirical software engineering (Oliner et al. 2012). They store data coming from different types of applications giving access to quantitative information about software, hardware, and systems performance in real operational environments.

System logs have been extensively used in diagnosing faults, scheduling applications and services of hardware (Featherstun and Fulp 2010; Fulp et al. 2008; Fu and Xu 2007), or in dependable computing, and in computer networks management and monitoring (Fu and Xu 2010, 2007; Yamanishi and Maruyama 2005; Steinder and Sethi 2004; Gross et al. 2002; Mannila et al. 1997). In software engineering, system logs have been employed to model the workflow design of the activity processes (van der Aalst et al. 2003), like Petri nets (Valette et al. 1989).

Oliner et al. (2012), overviewed some of the most common applications of log analysis discussing limitations and challenges. According to the authors, log analysis is usually performed to understand system performance, like estimating and predicting system failures. A typical approach to estimate system failures via log analysis is to identify abnormal system behavior against the system operational profile, which describes the system expected behavior (Oliner et al. 2012; Forrest et al. 1996; Oliner and Stearley 2007). As we cannot access to information on the operational profile (e.g., operational states) without disclosing protected information, we decided to describe the abnormal behaviors with sequence abstractions that include events in error state.

Oliner et al. (2012) have also warned researchers to use results of log analysis without considering the evolution of the system under study. Specifically, they say that "over the course of a system's lifetime, anything from software upgrades to minor configuration changes can drastically alter the meaning or character of the logs." This makes any log analysis harder to implement. In our work, we limited the influence of system evolution, by observing the system within a short period in which no specific upgrade, test, or drastic change happened.

To predict system failures, we follow the strategy of Munson and Khoshgoftaar (1992) that originally classifies software modules by their changes and, recently, has been used to classify defective software modules (Nagappan et al. 2010, 2008; Nagappan and Ball 2005).

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 (TP_r) is high) and raise few false alarms (false positive rate (FP_r) 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 classification 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 stress the relevance of an alarm..

Liang et al. (2007) code sets of events in a time window of given size by the counts of events at different severity levels (e.g., warning, error, etc.) and of different event textual description. Fulp et al. (2008) use two different abstractions: vector of multiplicities of different event types and spectrum-kernel representation introduced in Leslie et al. (2002) to describe the amino acid composition of proteins. In both abstractions, sequence length is not determined by the context and authors need to set up different experiments to compare SVM performance over sliding time-windows. Featherstun and Fulp (2010) use sliding time window also in combination with tag numbers.

Fronza et al. (2011, 2013) use context rules to determine the sequence length as in our study. Like in our case, they use system logs and break sequences by a specific event task (e.g., "log-in") or at the end of the day. They do not break sequences at the event in error state, though. Thus, sequences might contain further events after the error event. In Fronza et al. (2013), the item set of events is manipulated with Random Indexing (Sahlgren 2005) to encode time ordering in sequences. Fronza et al. (2011) predict defective sequences with models of survival analysis (Cox proportional hazard model) and SVMs (Linear and Radial Basis Function kernels). The use of the Cox model requires some technical assumptions that reduce the number of features available for the analysis.

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 $TP_r \geq 70\%$
Fulp et al. (2008)	Log sequences as vector of event (tag) multiplicities encoded in a given base (spectrum-kernel representation) or as in Liang et al., Liang et al. (2007). Target events are disk failures	One SVM, Linear kernel	Two Linux-based computer clusters. Half of the disk failure sets are randomly selected for training and the other half is used for testing. Hold-out repeated 100 times	Best results with spectrum-kernel representation: $TP_r = 75\%$ as and about $FP_r = 25\%$

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 three-months (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 tool[1].

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 logged-in users from some computers and recorded into 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

[1] http://www.r-project.org/

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

PCs and users. We formalize these concepts in Section 5.1.

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, classification, and cross-validation. The method is applied to every application A and with three types of classifiers: 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.

We chose 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 product-moment 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 non-linear 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 classifier 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 minimum-size 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. We adopt this triplet strategy and we additionally vary the construction of a splitting according to a parameter. We call it *parametric splitting*. With parametric splitting, the behavior of classifiers is observed over different triplets obtained from the same original data set. A coherent response of a classifier over the different triplets indicates the independency of the result from the specific splitting technique. We applied parametric splitting to two well-known techniques of splitting.

4.4 Classification, Section 8.

SVM is one of the top 10 data mining algorithms (Wu et al. 2007). SVM solves a mathematical optimization problem to find the separating hyper-plane that has largest distance to the nearest training data point of any classification group (functional margin between two groups).

In this study, we consider the basic Linear (L), the Radial Basis Function (RBF), and the Multilayer Perceptron (MP) learners. The three machines classify input for full and reduced feature spaces. For each feature space and cut-off, they are cross-validated on four triplets of sets obtained splitting the original data sets by sample balancing or incremental percentage splitting. The classification identifies model(s) that best fit and predict defective features for each cut-off value. We use true and false positive rates, balance, misclassification rate, and precision to compare classifiers. We discuss the relation among these measures specifically for imbalanced sets. Finally, the resulting performance is compared with the baseline literature (Table 9) in classification of system log sequences.

4.5 Cross-validation, Section 9.

The goal of cross-validation is to determine the best predictors. Cross-validation consists of model training and generalization. A model learns its weights and parameters on a training set (Training) and calculates its prediction performance on the new instances of a validation set (Generalization) (Bishop 1996).

To apply cross-validation, we need to have two samples from the same population, which in many cases are not available. When the original data set is large enough, a typical solution is to randomly split it. In this study, as we are using MP, we follow the approach in Khoshgoftaar et al. (1997) and Khoshgoftaar and Lanning (1995), and randomly split the original data sets into a *fit set* and a *validation set* and then randomly split the fit set into a *training set* and *test set*. The test set is used to determine the quality if fit of a trained classifier. At the end, we get a triplet of sets.

- **Training.** The model is trained on the training set to identify its internal structure. Typically, it is trained more times and evaluated against a loss function to determine the model weights corresponding that produce the mean absolute error. In our analysis, we perform the training on 500 epochs (an epoch is one pass through all training instances).
- **Quality of fit.** For a given training set and test set, a trained model is evaluated on the test set for quality of fit (accuracy in estimation). We call level of quality of fit the measure of this accuracy.
  - **Topology selection.** MP neural network has additional parameters that determine its topology. Number of hidden layers (hl), number of hidden nodes (h), momentum (m), and learning rate (l) define the topology. For a given topology and set pair, training and test set, the neural network is first trained on the training set and then evaluated on the test set. We repeat the procedure for all combinations of topology parameters. The model with the best misclassification rate defines the best topology.

  A *positive delta* indicates when one model outperforms the others in quality of fit for given training and test sets. The number of positive deltas over the different triplets of a splitting type measures the overall predominance of a model in quality of fit.
- **Generalization** The models with high quality of fit are then evaluated on the new instances of the validation set for generalization (accuracy of prediction).

5 Data Pre-Processing

5.1 Sequence Abstraction

One of the major challenges in log analysis is to define sequence abstraction as logs typically contain unstructured data. Although there are several proposals to

standardize logs, (Jiang et al. 2008), no specific standard has been extensively adopted. In this study, we follow the process of log standardization proposed in Jiang et al. (2008) and create sequences with the highest possible interpretability while keeping minimal knowledge and effort required to build them. Specifically, we inspected logs and interviewed system users to set sequence length and rules to determine it. Then we reviewed the information in logs and decided to categorize logs in sequences by their event type so that a sequence is a finite set of events of different types. Other studies use the longer event description to characterize such logs (Salfner et al. 2006). We find though that the description of the event task is synthetic (one or two words) and more standardized and, as such, requires less knowledge and effort to be parsed. What resulted is described in the following.

Let E_A be the set of all the events for a given application A. UE_A is the set of unique event types in A and μ_A its cardinality. For each application A, UE_A is uniquely defined. For example, in Table 4, there are six events and four unique event types, namely, {Log in, General, Performance, Systems}.

Table 4: Information used to build sequence abstractions 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.

In our system, only two-third of the applications share any event type and they share at most three unique event types (one is "Login"). Overall, we have found 125 distinct event types with average 6.4 event types per application and variation range 1-87 event types per application, Fig. 1.

The set of sequences S_A is a partition of E_A. The partition is defined by rules that identify sets of sequential events of the application A. The rules aim at determining the start and the end of a sequence. To determine the rules we performed a context analysis interviewing local experts. At the end, we came up with the following context rules:

- A sequence starts with an event of type "Log-in",
- A sequence starts with the beginning of the day, or

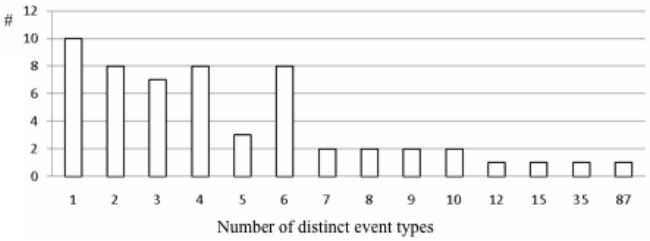


Fig. 1: Number of applications (y-axis) with a given number of event types in our system

- 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, s_1 and s_2. s_1 has two events of type "Log-in" and "Systems." s_2 has four events of three types {General, Log-in, 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

$$sv = [\mu_1, ..., \mu_{\mu_A}].$$

The multiplicities are entered in the vector according to the lexicographical order of event types in UE_A.

The vector sv is called *Sequence Type* of s.

This association defines the map into the μ_A-dimensional Cartesian product of natural numbers:

$$f : s \in S_A \rightarrow sv \in N^{\mu_A}$$

In Table 4, s_2 is mapped into the vector $[2, 1, 1, 0]$, when the unique event types are UE={General, Log-in, Performance, Systems}. The image under f defines the set of sequence types:

$\mathbf{ST_A} \subset N^{\mu_A}$, is the set of all the vectors $sv \in N^{\mu_A}$ for which there exists a sequence $s \in S_A$ such that $f(s) = sv$.

In Table 3, the vector $[2, 1, 1, 0]$ belongs to ST, as there exists s_2 that maps into it.

The inverse image $f^{-1}(sv)$ with $sv \in ST_A$ is the set of all the sequences that have type sv. For example, s_2 is in the inverse image of $sv = [2, 1, 1, 0]$. We define the *multiplicity* of a type sv the cardinality of its inverse image:

$$\mu(sv) = |f^{-1}(sv)|$$

As such,

the multiplicity $\mu(sv)$ of sv is the number of sequences that have sequence type sv.

For example, in Table 5, $sv = [2, 1, 1, 0]$ has multiplicity $\mu(sv) = 4$.

$\mathbf{MST_A} \subset ST_A$ is the set of sequence types with $\mu > 1$

A sequence is *defective* if it contains an event in error state. Defective sequence types are sequence types for which there exists at least one sequence that maps into them and has an event in error state. With this definition, a non-defective sequence might be mapped into a defective sequence type, as well. This means that there exists a combination of event types that lead to an error, but not all the combinations of the same event types do. In Table 4, s_2 is defective and $[2, 1, 1, 0]$ is the corresponding defective sequence type.

The number of errors, $\rho(sv)$, of a sequence type sv is the number of events in error state of all the sequences that map into it.

As such, a sequence type is defective if $\rho(sv) \geq 1$. A sequence type is *c-defective* if $\rho(sv) \geq c$.

$\mathbf{DefST_A} \subset ST_A$ is the set sequence types with $\rho \geq 1$.

In Table 4, s_2 is defective and $[2, 1, 1, 0]$ is 2-defective sequence type as $\rho([2, 1, 1, 0]) \geq 2$.

A sequence and a sequence type can be triggered by the use of one or more users. We denote with

$\nu(sv)$ the average number of users that have triggered events in sequences of type sv.

$\nu(sv) > 1$ indicates that at least one sequence of type sv have more than one user and sv is *distributed*.

$\mathbf{DistST_A} \subset ST_A$ is the set of distributed sequence types.

Given the set of unique operators of Table 4, a vector $sv = [2, 1, 1, 0]$, with $\mu(sv) = 4$, $\nu(sv) = 2.5$ can be generated by the sequences in Table 5.

Table 5: Example, $sv = [2, 1, 1, 0]$, $\mu(sv) = 4$, $\nu(sv) = 2.5$

Sequences	Users
(Log In, Performance, General, General)	{26210, 26210, 5776, 5776}
(Log In, General, General, Performance)	{5776, 5776, 26210, 26210}
(Log In, General, Performance, General)	{6601, 6601, 5776, 26210}
(Log In, General, General, Performance)	{6601, 6601, 5776, 3323}

5.1.2 Feature Spaces

As adding variables can only increase classifiers' performance (Guyon and Elisseeff 2003), with no loss of generality, we define the feature space of an application A as the $(\mu_A + 2)$-dimensional space of vectors formed by sequence types, sv, their multiplicity, $\mu(sv)$, and their average number of users, $\nu(sv)$:

$$\Im_A = \{v \in N^{\mu_A+2} | \exists sv \in SV_A : v = [sv, \mu(sv), \nu(sv)]\}$$

For example, the vector $t = [2, 1, 1, 0, 4, 2.5]$ in $\Im$ can be generated by the sequences in Table 5. A vector $v \in \Im$ is our new sequence abstraction that we use as input for the SVMs. The vector includes all the basic information about a sequence. We will check whether this information is redundant using IG (Quinlan 1987) and eventually reduce it.

5.2 Parametric classification categories

The output of a classifier depends on the definition of its categories. As mentioned, we propose to define categories by the number of errors, ρ. Classifying sequences by ρ might be complex, though, as this number does not have a specific upper bound. Namely, this infinite categorization can be so sophisticated and data can be so imbalanced and scarce that classifiers are not able to converge. A typical approach used in reliability analysis is to classify input into two categories only (Denaro and Pezzè 2002; Khoshgoftaar et al. 1997; Khoshgoftaar

and Lanning 1995; Porter and Selby 1990; Le Gall et al. 1990), and use a membership criterion to associate entities with categories. In our study, two parameterized disjoint groups define the output:

$G_1(c) = \{v \in \Im | \, \rho(sv) \geq c\}$ c-defective
$G_2(c) = \{v \in \Im | \, \rho(sv) < c\}$ not c-defective

Each value of the parameter c defines a new classification problem.

Note that $G_2(1)$ includes no defective sequences, whereas $G_2(> 1)$, in principle, might include defective sequences, i.e. sequence types that have more than zero and less than c errors. As we mentioned, the value of c needs to be chosen carefully as it changes the composition of G_1 and G_2 and the performance of the classifier. Setting the value of c too high might produce small G_1 on which the classifier is not able to give significant result. On the other hand, high values of c can be used to investigate the most problematic sequence types (Kim et al. 2011). In defining the value of c, one needs to trade off the technical performance of the classifier and the requirements of the given research problem. To give the idea of variation of the findings, we will discuss this trade-off at different values of c.

5.3 Descriptive Analysis of the Study Data Sets

In this section, we analyze the 50 sets of sequence types $\mathrm{ST}_i, i = \{1, ..., 50\}$, to eliminate those that have $\mathrm{G}_1(1) = \phi$ and understand their heterogeneity with respect to the number of errors, ρ, the average number of users, ν, and the multiplicity of sequence types, μ.

Table 6 and the box plots in Fig. 2 describe the distribution over applications of the size of ST_i (sequence types), MST_i (sequence types with multiplicity greater than one), DefST_i (defective sequence types), and DistST_i (distributed sequence types). In Table 6, we removed ST_{12} as it assumes values far higher than other data sets (ST=35828, MST=7419, DefST=16418, DistST=27205).

Table 6: Size of the sets, ST, DefST, DistST over applications. One outlier has been removed.

	ST	MST	DefST	DistST	DefST / ST
Mean	72	13.21	41.23	47.25	21%
SD	125.63	21.77	111.77	118.72	30%
Max	783	109	687	768	88%
Min	1	0	0	0	0 %

Notice that as in typical studies in software engineering (Menzies et al. 2007a), defective sequence types are under represented (mean(DefST/ST)=21% ± 30%).

Examining the box plots in Fig. 2, we see that all distributions are non-parametric and with few outliers. The size of STs mainly varies between 10 to 60 sequence types. We considered large sets the ones with more than 100 items.

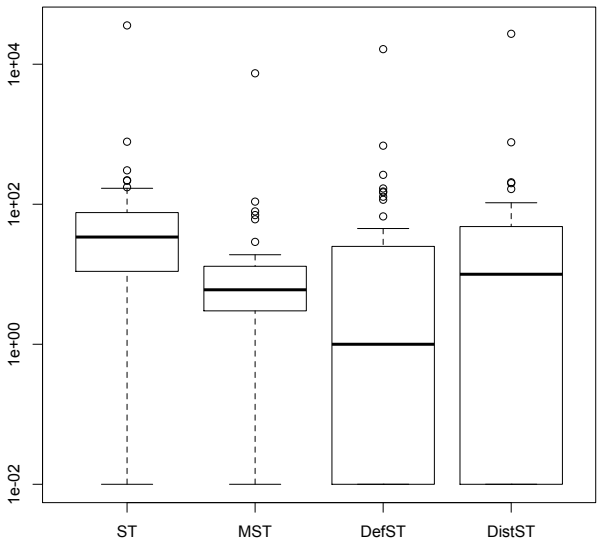


Fig. 2: Box plots. Size distribution of ST, MST, DefST, and DistST over applications. Log scale. 0 values set to 0.01

Half of the sets have no defective sequence types. These applications are excluded from our analysis reducing the sets to *25 sets*. Half of the sets have more than 50 sequence types with $\mu > 1$ or sequences triggered by less than five users.

A non-parametric (Spearman) correlation at 0.05 significance further shows that the size of ST, DefST, and DistST are mutually, significantly correlated (with all values around 0.99). These correlations stress the relevance of large size ST_i in which both defective and distributed sequence types are numerous.

5.3.1 c-Defective Sequence Types

In this section, we examine the sets ST_i by different values of cut-off, c. We limited the analysis to $c \leq 4$ as sequences types with more than five events in error state are rather rare in our data (only seven data sets have more than 5% defective sequence types and only one has more than 15%).

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 mainly have $c = 1$.

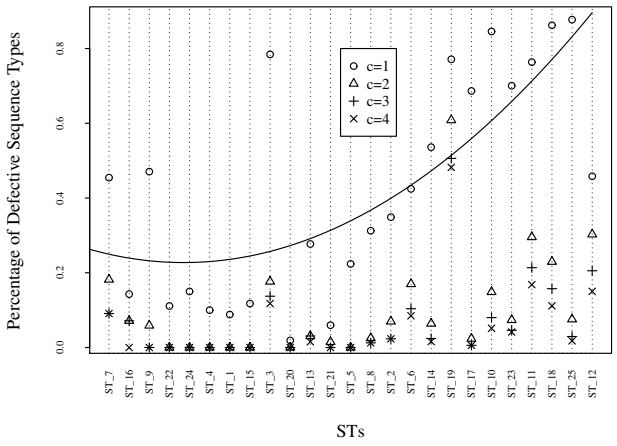


Fig. 3: Percentages of c-defective sequence types over applications. c=1,...,4. ST_i are ordered by their size.

Altogether, we can see that there is a great heterogeneity among the data sets for which any result across applications or values of c would be of great value. We further note that increasing the value of c increments the imbalance of the sequence types over the classification categories drastically lowering the percentage of c-defective sequences for $c = 2$. This imbalance must be taken into account in the discussion of classifiers' performance (Menzies et al. 2007b; Zhang and Zhang 2007; Menzies et al. 2007a). The relation between size and percentage of defective sequence types appears to be more than quadratic in our system, Fig. 3.

6 Feature reduction

We use IG to rank attributes arranging them from the most to least informative. As illustrated by Menzies et al. (2007b), IG measures the reduction in entropy of one class distribution C achieved by conditioning it with a new variable X:

$$\mathrm{IG}(X) = H(C) - H(C|X)$$

where H() is the expected mean of number of bits required to encode C:

$$H(C) = -\sum_{o \in C} p(o) log_2 p(o)$$

For each data set, the initial attributes consist of the event multiplicities μ_i, the sequence multiplicity μ, and sequence distribution ν. For each application, we consider all the attributes that contribute to the information of a class distribution C determined by the number of errors in sequence types:

$$C = \{[sv, \mu(sv), \nu(sv)] : \rho(sv) \geq c\}$$

Table 7 illustrates the attributes selected by IG per applications and value of c. "Orig. Attr. #" indicates the length of the vector $[sv, \mu(sv), \nu(sv)]$ and "Red. Attr. #" the length of the vector after applying IG. Among the 25 data sets, seven data sets (all small size) are not reported in the table, as there is no information reduction for any value of c; for the same reason, for some data sets in the table, specific values are not reported ("-"). The table includes all large data sets and half of small data sets. * and ** indicate whether respectively μ or ν and μ are included.

The reduction of event types is significant in 18 out of 25 data sets for at least one value of c. This indicates that the information contained in sequences is redundant for the majority of the applications. In particular, small data sets, typically representing applications focused on specific task (like ERP modules), have fewer original attributes, but proportionally more redundancy. a. In all but five feature spaces, sequences containing more than one event type contribute to the error classification problem. This support the findings in Jiang et al. (2009a) for which "a failure message alone is a poor indicator of root cause, and that combining failure messages with multiple log events can improve low-level root cause prediction by a factor of three.? This is also supported by the fact that in our data

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%

sets error types are labeled with generic description as "Generic? or "Generic Error.? Worth noticing that μ is a key attribute (i.e., not reduced) when $c > 1$.

7 Parametric Splitting

We use two types of sample manipulation: sample balancing and incremental percentage splitting. In both cases, splitting is parametric, in that different parameter values define different splitting percentages. All samplings are random, uniform without replacement.

Incremental percentage splitting (t-splitting) randomly splits data sets into fit set and validation set with parametric increasing proportion and then randomly splits fit sets into training and test sets in a fixed proportion. For a splitting parameter t :

- Validation set: $t*100\%$ random selection of the original data sets,
- Fit set: complement of the validation set in the original data sets,
- Test set: 30% random selection of the fit set,
- Training set: complement of the test set in the fit set.

The output of incremental percentage splitting repeated for $t = 1/2, 1/3, 1/4, 1/5$ consists of four triplets of sets (Training, Test and Validation set) obtained as above Incremental percentage splitting helps understand whether set size influences classifiers' performance. Similar results over t values indicate no influence of size on findings.

Sample Balancing (k-splitting) builds training sets for classifiers that include a balanced representative of all the output classes of interest. We also ensure that for every choice of $c : 1)$ the two groups $G_1(c)$ and $G_2(c)$ are equally represented and 2) sequence types with zero faults are always present in $G_2(c)$:

- Validation set: $1/3 * 100\%$ random selection of the original data set,
- Fit set: complement of the validation set in the original data set,
- Test set: 30% random selection of the fit set,
- Training set: all the c-defective sequence types in the Fit set minus the test set (cardinality = r), plus k randomly selected not c-defective sequence types that have zero errors, plus $r - k$ randomly selected not c-defective sequence types with more than zero errors. One example of this splitting method can be found by Khoshgoftaar et al. (1997): with $k = 5$.

The output of k-splitting repeated for $k = 2, 3, 4, 5$ consists of four triplets of sets (Training, Test and Validation set) obtained as above. In total, over applications, splitting parameter, and values of c, we potentially have 400 (=25*4*4) triplets for each type of splitting, but not all the triplets can be used. In both types of splitting, we designed an algorithm that checks if, for a given value of c and a given value of k, the sets in a triplet have enough items to be used with classifiers. If one of the cases happens, the algorithm returns and the corresponding triplet is excluded from the analysis, Fig. 4.

```
# number of original data sets with fixed value of c
APPLICATIONS = 25
for i in APPLICATIONS
    # Feature space of data set ST_i
    DATA = {sv}
    # number of errors of vector sv
    ρ
    DEFDATA= select 'Defective' in DATA
    NONDEFDATA= sv ! in DEFDATA
    NONDEFZERODATA= sv in NONDEFDATA && ρ(sv) = 0
    # not enough c-defective sequence types
    if card(DEFDATA)< k ||
    # not enough not c-defective sequence types with ρ(sv) = 0
    card(NONDEFZERODATA)< k ||
    # not enough not c-defective sequence types with ρ(sv) > 0
    card(NONDEFEDATA - NONDEFZERODATA)< k ||
    # not enough not c-defective sequence types with ρ(sv) > 0
    # to balance data set
    card(NONDEFDATA - NONDEFZERODATA)< −k+card(DEFDATA)
       # the data set is excluded
       return i else
    # the data set is included
    return φ
```

Fig. 4: The algorithm that identifies usable fit data sets

8 Classification

We classify features by c-defectiveness using Support Vector Machines (SVMs (Vapnik 1995)). A SVM separates training instances by a hyper-plane defined by its support, i.e., a vector perpendicular to the plan w and the plane's distance from the origin b. A new instance z is classified according to which side of the hyper-plane it belongs to. When instances are not linearly separable, the SVM algorithm maps instances to a high-dimensional feature space. In this feature space, a separating hyper-plane can be found such that, when projected back to the original feature space, it describes a non-linear decision function. The projection is performed by a kernel function that expresses the inner product between two instances in the original feature space, (the *Kernel trick*). In our work, we use the following three kernel functions:

K(x, w)= $x \cdot w + b$, basic linear (L)

K(x, w)=exp $(-||x - w||^2)$, radial basis function (RBF)

K(x, w)= tanh$(\alpha x \cdot w + b)$, multilayer perceptron (MP)

Varying the support, we get different instances of SVMs. The best performance on training instances identifies the support of SVM that best fit our data. For each instance in the test set, the SVM obtains a probability value (for each group a value from 0 to 1). Then a new instance is predicted as c-defective, i.e., in $G_1(c)$, if the classifier's output probability is greater than 0.5. The output is matched against the expected distribution obtained with the a priori classification (the expected group has 1, the other 0) to get the SVM quality of fit. Finally, the best performing SVMs over the test sets are tested for prediction accuracy on the validation instances. Again the classifier's probability is matched against the expected distribution over the categories.

8.1 Measures of classifier's performance

An output is Positive if it is in $G_1(c)$, it is Negative if it is in $G_2(c)$. True Positives (TP) or True Negatives (TN) are, respectively, the number of actual c-defective or actual not c-defective features that have been correctly classified. False Positives (FP) or False Negative (FN) are respectively the number of actual not c-defective or actual c-defective sequence types that have been misclassified.

To assess classifier performance, we use the measures in Table 8. MR, FP_r, TP_r, and precision are expressed in percentages or equivalently with numbers between 0 and 1. Balance is a number between 0 and 1. Best classifiers have low MR and FP_r and high TP_r, balance, and precision. Neg/Pos can be any non-negative number.

Comparison of classifiers with one single measure does not work. For example, low values of MR do not prevent high FP$_r$, (Menzies et al. 2007b). Literature typically uses three measures TP$_r$, FP$_r$, and precision or a combination of them, but in fact comparing different classifiers on the same set or on sets with similar Neg/Pos ratio is a problem of two degrees of freedom if the ratio is known. In other words, just two of these measures suffice. This is proved by the Conversion Formula introduced by Zhang and Zhang (2007), Table 8.

The Conversion Formula also shows that classification on sets with very high Neg/Pos ratio cannot easily achieve high precision even for relatively small values of FP$_r$, Menzies et al. (2007b) and ZhangZhang2007. For example, Menzies et al. (2007b) obtained TP$_r$ = 71% and FP$_r$ = 25% (balance 0.73) on average across eighth data sets with average Neg/Pos= 12.8. Using the Conversion Formula on each data set, the average precision only equals to 26%. There is no way to increase this value but reducing the Neg/Pos ratio, i.e. balancing the data sets. Menzies et al. (2007b) choose TP$_r$ and FP$_r$ and further use balance, an operational measure to evaluate the distance of (TP_r, FP_r) from the ideal pair (100%, 0%). At the ideal pair, the classifiers identify all positive values $TP_r = 100\%$ with no false alarm $FP_r = 0\%$. Higher balance values fall closer to the ideal pair.

Table 8: Measures of classifier performance, (Bishop 1996; Menzies et al. 2007b)

Name	Formula
Misclassification rate, (MR)	$\frac{FP+FN}{FP+FN+TP+TN}$
True Positive rate, (TP_r)	$\frac{TP}{FN+TP}$
False Positive rate, (FP_r)	$\frac{FP}{FP+TN}$
Balance	$1-\frac{\sqrt{(TP_r-1)^2+(FP_r)^2}}{\sqrt{2}}$
Precision	$\frac{TP}{FP+TP}$

Conversion Formula, Zhang and Zhang (2007)

$$\text{Precision}=\frac{1}{1+\frac{FP_r}{TP_r}\cdot\frac{Neg}{Pos}},$$

Neg=TN+FP

Pos=TP+FN

Table 9 shows FP_r and TP_r and precision for the baseline studies. These studies use SVMs, but different sequence abstractions. None of them uses MP. Table 10 compares studies that use different predictors and approaches to learn errors from system logs. Studies that do not use or for which we cannot derive TP_r FP_r and precision are not included. In both tables, in case the value Neg/Pos is available, we use the Conversion Formula to eventually compute the measures missing in the original papers.

Table 9: Best SVMs and their prediction performance for the baseline studies. In case of multiple data sets, values are averaged. * indicates values computed with the Conversion Formula. ^{w} stands for "weighted".

Study	SVM	FP_r	TP_r	Prec.	*Neg/Pos*
Fronza et al. (2011)	L & RBF	1.7%	42%	60%*	39
Fronza et al. (2013)	L^w & RBF^w	16%	65.5%	50%*	39
Liang et al. (2007)	RBF	NA	$\approx 85\%$	$\approx 55\%$	NA
Fulp et al. (2008)	L	NA	$\approx 75\%$	$\approx 70\%$	NA

Table 10: Best classifiers and their prediction performance for studies that use other models. In case of multiple data sets or resampling, values are averaged. * indicates values computed with the Conversion Formula.

Study	Model	FP_r	TP_r	Prec.	*Neg/Pos*
Vilalta and Ma (2002)	Ass. rules	4%	39%	99%*	0.03
Fronza et al. (2011)	Cox mod.	27%	59.3%	18%*	39
Salfner et al. (2006)	Heur.	1%*	92.3%	80%	28
Jiang et al. (2008)	Heur.	NA	90%	93%	NA
Shang et al. (2013)	Heur.	NA	NA	33.7%	NA

9 Cross Validation

9.1 Training

Models are trained on training sets per splitting type. The classification on training sets is repeated for every choice of c. In the specific case of the MP classifier, training is also performed for every choice of the topology parameters.

9.1.1 Topology Selection for MPs

We vary the topology parameters h between 1 and 10, l and m in $\{1, 1/2, ..., 1/10\}$, and fix hl equal to one. The best topology corresponds to the smallest MR. Topology selection is replicated for $c = 1, \ldots, 4$, $t = 1/2, \ldots, 1/5$, and $k = 2, \ldots, 5$, on full and reduced feature spaces, and for all feature spaces.

Fig. 8 in the Appendix shows the best topology parameters h, m, and l (x-axis) across the applications (y-axis) in the case of $c = 2$ and $k = 2$, for k-splitting, and $c = 2$ and $t = 1/3$, for t-splitting, using the full feature sets. The z-axis reports the values of the parameters.

We were not able to determine the topology on full or reduced feature spaces for all applications. In the case of t-splitting, this is because there are not enough errors in the fit sets. In the case of k-splitting, this is for the insufficient number of zero errors or not c-defective features. In addition, because of the large size of ST_{12} and computational memory limitation of the R tool, we were not able to determine topology and performance of any classifier on this data set.

9.2 Quality of fit

To have a first idea of the overall performance, we initially assess quality of fit with MR on test sets for each value of c, type of splitting, and classifier. Then we use positive delta to count classifiers that outperform the others, as described in Section 4.5. For each value of c, Table 11 illustrates the results.

Table 11: Number of applications for which a classifier outperforms the others in quality of fit

		c=1	c=2	c=3	c=4
t-splitting Full	MP	16	8	7	3
	RBF	1	0	0	0
	L	12	6	3	7
t-splitting Reduced	MP	11	8	7	5
	RBF	1	1	1	1
	L	4	3	4	3
k-splitting	MP	5	2	1	2
	RBF	1	2	1	0
	L	5	7	3	2

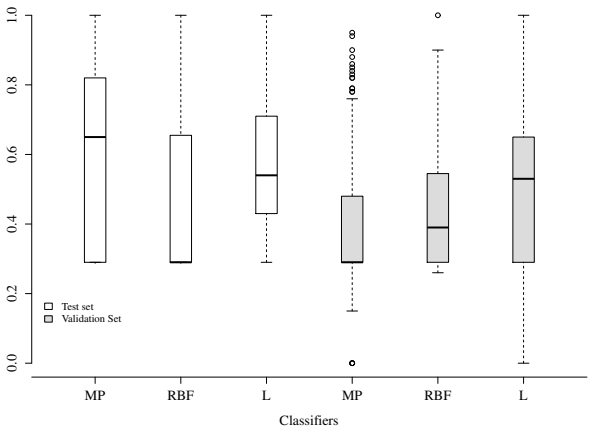
Best classifiers have the greatest positive deltas over all eight pairwise comparisons obtained from a splitting type. Table 11 reports the number of applications for which a given model is the best classifier. In some cases, more than one model evenly outperform the others. The numbers show that for the majority of the applications, MP outperforms the other two models. L outperforms the other classifiers for greater values of c and k-splitting.

9.3 Generalization

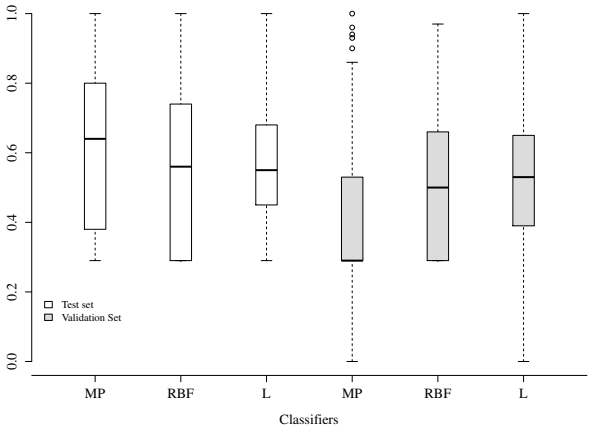
We base our strategy to assess prediction performance on four measures - balance, TP_r, FP_r, and precision. Fig 5 plots the balance distribution of the three SVMs on both test and validation sets for full, reduced, balanced data sets. The distributions are defined over different t-splitting triplets, and four cut-off values.

We use balance to select models with a given level of quality of fit (on test sets) and discuss the generalization performance (on validation sets) of the resulting classifiers with TP_r, FP_r, and precision. As a matter of example, we set balance greater than 0.73 as for the level of quality of fit. This threshold was found by Menzies et al. (2007b) for fault classification and is close to the reference values of data stored in the University of California Irvine machine learning database[2]. In addition, the threshold corresponds to the highest value found in the baseline studies. We chose this high value

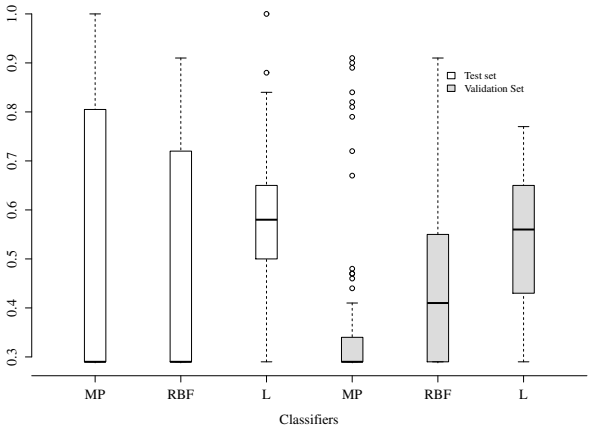
[2] http://archive.ics.uci.edu/ml/



(a) Balance of models on full feature spaces



(b) Balance of models on reduced feature spaces



(c) Balance of models on balanced feature spaces

Fig. 5: Balance distribution of the three classifiers on test (left) and validation (right) sets.

to show that we can have high quality of fit (i.e., above the threshold) and prediction good on average and superior to existing literature on single applications. An high quality of fit ensures that predictions are made on the actual system behavior. Fig. 5 shows that L outperforms the best both quality of fit and generalization. After filtering for high quality of fit, we will see that L will not be the best predictor anymore.

Finally, we repeat the analysis for full and reduced feature spaces with t-splitting and for full feature spaces with k-splitting, as described in the next section.

10 Findings

In this section, we investigate prediction performance of SVMs varying the t-splitting parameter t, before (Full feature spaces) and after feature reduction (Reduced feature spaces), and varying the k-splitting parameter k, (Balanced feature spaces).

Full feature spaces. Table 12 describes the average performance in quality of fit and generalization of the classifiers that have balance greater than 0.73 on test sets. We were able to find such classifiers for 20 applications.

Table 12: Full Feature Spaces. Average performance of best quality of fit classifiers across values of c and over 20 applications

	test			validation		
	balance	FP_r	TP_r	balance	FP_r	TP_r
c=1	0.89	0.08	0.89	0.4	0.3	0.38
c=2	0.91	0.02	0.88	0.48	0.11	0.35
c=3	0.88	0.01	0.84	**0.55**	**0.05**	**0.39**
c=4	0.94	0.01	0.92	0.51	0.03	0.33
average	0.90	0.03	0.88	0.49	0.12	0.36

To obtain a row of the table, we average the scores (both for quality of fit and generalization) over splitting parameter values, applications, and classifiers with balance greater than 0.73 on the test set. It is worth noticing that, on average, performance is moderately low for generalization. This indicates that some classifiers although fitting very well our data are not excellent predictors on some splitting triplets.

To understand the performance of classifiers on single triplets, we expanded Table 12 into Table 13. To obtain a row of Table 13, we first select all classifiers with balance greater than 0.73 on the four test sets obtained varying t. For each value of t, we then choose the model, X, among L, MP, and RBF that has the greatest balance value on the test set. Then we filter out the splitting triplets that have X with balance less than average, i.e., less then 0.49, on validation sets. Table 13 shows the remaining cases, i.e., the best model for quality of fit that has balance greater than 0.73 on test set and 0.49 on validation set. Notice that we did not find such model for all t values or applications.

Table 13: Full Feature spaces. Best classifiers and their performance across values of c and triplets of t-splitting.

			test			validation		
	ID	SVM	bal.	FP_r	TP_r	prec.	FP_r	TP_r
c=1	ST_2	MP	0.81	0.00	0.73	0.63	0.25	0.53
	ST_2	L	0.76	0.08	0.67	0.75	0.19	0.75
	ST_2	L	0.77	0.12	0.70	1.00	0.00	0.71
	ST_2	L	0.75	0.14	0.68	0.76	0.08	0.60
	ST_6	MP	0.88	0.15	0.92	1.00	0.00	0.33
	ST_6	MP	0.88	0.11	0.87	0.67	0.07	0.29
	ST_6	MP	0.89	0.05	0.86	0.80	0.17	0.71
	ST_9	MP	1.00	0.00	1.00	0.50	0.20	0.33
	ST_{11}	RBF	0.83	0.17	0.83	0.89	0.43	0.91
	ST_{11}	RBF	0.79	0.25	0.85	0.96	0.10	0.87
	ST_{11}	RBF	0.86	0.13	0.85	0.88	0.42	0.86
	ST_{13}	MP	0.92	0.03	0.89	0.67	0.15	0.50
	ST_{14}	MP	0.78	0.26	0.82	0.64	0.47	0.73
	ST_{15}	L	1.00	0.00	1.00	0.49	0.07	0.50
	ST_{21}	L	1.00	0.00	1.00	1.00	0.00	1.00
avg.			**0.86**	**0.10**	**0.84**	**0.78**	**0.17**	**0.64**
c=2	ST_{10}	MP	0.96	0.02	0.94	1.00	0.00	0.50
	ST_{11}	MP	0.86	0.06	0.81	0.85	0.04	0.50
	ST_{11}	MP	0.77	0.00	0.68	0.74	0.1	0.58
	ST_{11}	MP	0.81	0.03	0.74	0.57	0.31	0.57
	ST_{11}	MP	0.81	0.02	0.73	0.77	0.22	0.81
	ST_{17}	L	1.00	0.00	1.00	1.00	0.00	1.00
	ST_{18}	MP	0.92	0.00	0.88	0.89	0.04	0.95
	ST_{18}	MP	0.82	0.00	0.74	0.82	0.04	0.58
	ST_{18}	MP	0.96	0.02	0.94	0.87	0.03	0.75
	ST_{18}	MP	0.96	0.03	0.96	0.85	0.05	0.71
avg.			**0.89**	**0.02**	**0.84**	**0.83**	**0.08**	**0.60**
c=3	ST_{11}	MP	0.78	0	0.69	0.89	0.02	0.60
	ST_{11}	MP	0.87	0	0.82	1	0	0.69
	ST_{11}	RBF	0.84	0.12	0.8	0.86	0.03	0.73
	ST_{11}	RBF	0.81	0.09	0.74	0.68	0.13	0.75
	ST_{14}	L	1.00	0.00	1.00	1.00	0.00	0.50
	ST_{18}	MP	0.94	0.00	0.92	1.00	0.00	0.74
	ST_{18}	MP	1.00	0.00	1.00	1.00	0.00	0.65
	ST_{18}	MP	0.96	0.00	0.94	1.00	0.00	0.79
	ST_{18}	MP	0.94	0.00	0.91	1.00	0.00	0.86
	ST_{19}	MP	0.85	0.03	0.79	0.67	0.38	0.80
	ST_{19}	MP	0.79	0.00	0.70	0.69	0.52	0.97
	ST_{19}	MP	0.79	0.02	0.71	0.72	0.35	0.86
	ST_{19}	MP	0.81	0.05	0.74	0.75	0.24	0.75
avg.			**0.88**	**0.02**	**0.83**	**0.87**	**0.13**	**0.75**
c=4	ST_3	RBF	1.00	0.00	1.00	0.51	0.15	0.50
	ST_{11}	MP	0.81	0.00	0.73	1.00	0.00	0.53
	ST_{11}	RBF	0.75	0.03	0.65	1.00	0.00	0.29
	ST_{11}	RBF	0.84	0.04	0.78	0.89	0.02	0.70
	ST_{18}	MP	0.92	0.00	0.88	1.00	0.00	0.83
	ST_{18}	MP	1.00	0.00	1.00	1.00	0.00	0.47
	ST_{18}	MP	0.96	0.00	0.95	1.00	0.00	0.92
	ST_{18}	MP	0.97	0.00	0.96	1.00	0.00	0.43
	ST_{19}	MP	0.74	0.00	0.63	0.95	0.05	0.78
	ST_{19}	MP	0.93	0.05	0.92	0.84	0.21	0.87
	ST_{19}	MP	0.82	0.02	0.75	0.81	0.10	0.45
	ST_{19}	MP	0.85	0.03	0.79	0.62	0.38	0.59
avg.			**0.88**	**0.01**	**0.84**	**0.88**	**0.08**	**0.61**

On average, the resulting models have precision much higher than the ones found in the baseline studies that use SVMs, Table 9. For example, for $c = 1$, TP_r and FP_r are similar to the value found by Fronza et al. (2013) for weighted SVMs.

Overall, the table shows that across the values of c, MP outperforms RBF and L. In addition, notice that in the majority of the cases, MP or RFB consistently perform the best across different splitting sizes of the same application. This indicates stability of such predictors.

Reduced feature spaces. As for full feature spaces, Table 14 describes the average performance in quality of fit and generalization of the classifiers that have balance greater than 0.73 on test sets. Again, we were able to find such classifiers for 20 applications. The average balance value on the validation sets has a bit increased in comparison with full feature spaces. In particular, the best value is 0.57 and is taken for $c = 3$. The reason for this increase is the lower value of the false positive rate, FP_r. This is specifically visible for $c = 1$.

For each value of c, the values are averaged over about 30 cases, i.e. the ones for which we found classifiers that converge and have balance greater than 0.73 on test sets. Therefore, the moderate values of FP_r and TP_r suggest that for some choices of the t value, best fit models are not excellent predictors.

Table 14: Reduced feature set. Average performance of best quality of fit classifiers across values of c and over 20 applications

	test			validation		
	balance	FP_r	TP_r	balance	FP_r	TP_r
c=1	0.84	0.13	0.84	0.53	0.18	0.36
c=2	0.87	0.04	0.83	0.49	0.03	0.31
c=3	0.88	0.02	0.83	**0.57**	**0.04**	**0.41**
c=4	0.89	0.01	0.85	0.46	0.02	0.25
average	0.87	0.05	0.84	0.51	0.07	0.33

To understand for which feature sets this holds true, we expand Table 14 into Table 15. Table 15 shows that precision and false positive rates increase with feature reduction. For $c = 1$, precision overcomes any value found with SVMs in baseline studies (Table 9), and approaches the best value found in baseline studies that use other models (Table 10). In such last case, there are two studies that overall achieve results comparable with ours for $c = 1$. Vilalta and Ma (2002) found better precision (0.99), but much lower TP_r (0.39) and slightly better FP_r (0.04). Salfner et al. (2006) achieved better TP_r (92,3%) and FP_r (1%), but worse precision (80%).

Overall, as for comparison of Table 13 and 15 we see that:

Table 15: Reduced feature spaces. Best classifiers and their performance across values of c and triplets of t-splitting.

			test			validation		
	ID	SVM	bal.	FP_r	TP_r	prec.	FP_r	TP_r
c=1	ST_2	MP	0.83	0.10	0.79	0.82	0.06	0.36
	ST_3	RBF	0.77	0.29	0.85	0.90	0.25	0.92
	ST_6	MP	0.94	0.07	0.96	0.85	0.12	0.80
	ST_6	MP	0.89	0.05	0.86	0.87	0.11	0.82
	ST_6	MP	0.9	0.09	0.88	0.65	0.13	0.82
	ST_6	MP	0.89	0.06	0.86	0.82	0.08	0.78
	ST_{11}	RBF	0.82	0.19	0.83	0.94	0.20	0.86
	ST_{11}	RBF	0.85	0.16	0.86	0.92	0.19	0.83
	ST_{11}	RBF	0.82	0.20	0.84	0.89	0.38	0.89
	ST_{11}	RBF	0.77	0.28	0.84	0.96	0.22	0.91
	ST_{14}	RBF	0.75	0.22	0.72	0.67	0.42	0.58
	ST_{21}	L	1.00	0.00	1.00	1.00	0.00	1.00
	ST_{21}	L	1.00	0.00	1.00	1.00	0.00	1.00
avg.			**0.86**	**0.13**	**0.87**	**0.87**	**0.17**	**0.81**
c=2	ST_{11}	RBF	0.78	0.12	0.71	0.75	0.12	0.75
	ST_{11}	RBF	0.85	0.16	0.85	0.92	0.04	0.47
	ST_{14}	RBF	1	0.00	1.00	1.00	0.00	0.50
	ST_{18}	MP	0.89	0.04	0.85	0.97	0.01	0.80
	ST_{18}	MP	0.96	0.03	0.94	1.00	0.00	0.72
	ST_{18}	MP	0.87	0.03	0.82	0.93	0.02	0.86
	ST_{18}	MP	0.87	0.02	0.81	1.00	0.00	0.75
	ST_{19}	MP	0.77	0.06	0.68	0.81	0.50	0.98
	ST_{23}	MP	0.87	0.00	0.82	1.00	0.00	0.40
avg.			**0.87**	**0.05**	**0.83**	**0.93**	**0.08**	**0.69**
c=3	ST_3	RBF	1.00	0.00	1.00	1.00	0.00	0.40
	ST_{11}	MP	0.82	0.00	0.74	1.00	0.00	0.50
	ST_{11}	RBF	0.82	0.13	0.78	0.79	0.05	0.70
	ST_{11}	RBF	0.8	0.11	0.74	0.73	0.10	0.85
	ST_{11}	RBF	0.83	0.12	0.78	0.75	0.09	0.70
	ST_{18}	MP	0.98	0.00	0.97	0.95	0.01	0.94
	ST_{18}	MP	0.92	0.00	0.89	1.00	0.00	0.90
	ST_{18}	MP	0.93	0.00	0.89	1.00	0.00	0.80
	ST_{18}	MP	1.00	0.00	1.00	1.00	0.00	0.33
	ST_{19}	MP	0.73	0.02	0.62	0.92	0.07	0.81
	ST_{19}	MP	0.82	0.04	0.75	1.00	0.00	0.71
	ST_{19}	MP	0.83	0.02	0.75	0.61	0.56	0.83
	ST_{19}	MP	0.89	0.03	0.85	1.00	0.00	0.38
	ST_{23}	RBF	0.76	0.00	0.67	1.00	0.00	0.50
avg.			**0.87**	**0.03**	**0.82**	**0.91**	**0.06**	**0.67**
c=4	ST_3	L	0.76	0.00	0.67	1.00	0.00	0.33
	ST_{11}	MP	0.75	0.00	0.65	1.00	0.00	0.33
	ST_{11}	RBF	0.82	0.00	0.75	0.83	0.03	0.69
	ST_{18}	MP	0.97	0.00	0.96	1.00	0.00	0.55
	ST_{19}	MP	0.76	0.00	0.67	1.00	0.00	0.68
	ST_{19}	MP	0.80	0.22	0.82	1.00	0.00	0.63
	ST_{19}	MP	0.80	0.00	0.71	0.88	0.10	0.76
	ST_{19}	MP	0.83	0.01	0.76	0.78	0.20	0.67
avg.			**0.81**	**0.03**	**0.75**	**0.94**	**0.04**	**0.58**

- Best models for quality of fit are not always the best predictors for all splitting sizes of a feature set. Researchers must consider to replicate their study over different splitting sizes.
- After feature reduction, best models for quality of fit that have generalization performance more than average, have higher precision and true positive rate

although slightly worse false positive rate as predictors.

- Specially after feature reduction, prediction is solved by non linear SVMs: MP and RBF are better predictors than L.
- After feature reduction, a single classifier outperforms the others across splitting triplets (independence from the splitting size) and values of c (independence from the cut-off). Specifically, MP is a more stable predictor and performs better than RBF for more applications and across c values. On the other side, when we computed the Neg/Pos ratio of the validation sets over splitting triplets, we noticed that RBF is more resilient after feature reduction across different splitting sizes of the same feature set. For example,

	size	Pos	c		Neg/Pos val.
ST_{11}	220	68	1	RBF	0.27
			2	RBF	1.56
			3	RBF	3.35
			4	RBF	5.25
ST_{19}	166	128	1	-	0.23
			2	MP	0.96
			3	MP	0.89
			4	MP	0.89

  This result confirms the findings in Finan et al. (1996) that show as RBF is more resilient than MP on imbalanced training sets.
- Even if we require high quality of fit, SVMs achieve better prediction than the one reported in existing literature and approach precision of other types of models exiting in literature having on average similar generalization performance. These results will be further discussed in Section 11.1.
- After feature reduction, precision becomes much higher and increases with c values. The same holds true for FP_r that becomes very low with c values. In parallel, TP_r which is very high for $c = 1$ decreases when c increases.

Balanced Feature Spaces. Table 16 reports the generalization performance at individual applications after sample balancing. Notice that the linear classifier L is not present anymore. Precision increases significantly reaching the top values found by Vilalta and Ma (2002), but achieving much better performance. Overall, the k-splitting does not help identify a single classifier (e.g., ST_{11}), but it helps increase convergence for those data sets on which IG does not reduce the feature space (e.g., $c = 1$, ST_{10} and ST_{18}) and increase precision and generalization performance.

Worth noticing that when we artificially balance data, MP and RBF are equally represented.

Table 16: Sample balance splitting. Best classifiers and their performance across values of c and k-splitting triplets

			test			validation		
	ID	SVM	bal.	FP_r	TP_r	prec.	FP_r	TP_r
c=1	ST_{18}	MP	0.97	0.01	0.96	0.99	0.05	0.75
		MP	0.97	0.02	0.96	0.99	0.09	0.75
		MP	0.92	0.04	0.89	0.99	0.03	0.71
avg.			**0.95**	**0.02**	**0.94**	**0.99**	**0.06**	**0.74**
c=2	ST_{11}	MP	0.86	0.11	0.83	0.96	0.10	0.88
		MP	0.86	0.17	0.9	0.95	0.12	0.96
		RBF	0.82	0.23	0.88	0.95	0.12	0.96
		MP	0.85	0.18	0.88	0.95	0.12	0.92
	ST_2	RBF	0.82	0.07	0.75	0.88	0.07	0.67
avg.			**0.84**	**0.15**	**0.85**	**0.94**	**0.11**	**0.88**
c=3	ST_{11}	RBF	0.82	0.23	0.88	0.99	0.02	0.87
		RBF	0.82	0.19	0.84	0.97	0.07	0.73
	ST_{18}	MP	1.00	0.00	1.00	1.00	0.00	0.53
avg.			**0.88**	**0.14**	**0.91**	**0.98**	**0.03**	**0.71**
c=4	ST_{10}	MP	0.86	0.06	0.81	1.00	0.02	0.6
	ST_{11}	RBF	0.85	0.16	0.87	1.00	0.00	0.29
		RBF	0.86	0.15	0.87	0.97	0.02	0.29
		RBF	0.88	0.11	0.87	1.00	0.00	0.29
		RBF	0.91	0.08	0.91	1.00	0.00	0.29
avg.			**0.87**	**0.11**	**0.87**	**0.99**	**0.01**	**0.35**

11 Discussion

In Section 10, we have seen that results are better and more stable after feature reduction and sample balancing only improves prediction performance for some specific feature sets. As such, this section will discuss findings obtained after feature reduction and with t-splitting. Prediction performance with k-splitting will be used whenever we are able to provide better values.

We start comparing our results with baseline studies, Table 9 and 10. With our method, prediction performance is obtained from models with high quality of fit (balance greater than 0.73 on test sets). Selecting model with increasing quality of fit decreases the number of available models for prediction. Fewer models provide equal or lower prediction performance values. What we are going to demonstrate in this section is that even with high quality of fit, we are able to produce prediction performance that is superior to existing literature, specially when literature uses SVMs.

We then illustrate an example of how to interpret our numerical findings. We answer the business and research questions thereafter. Finally, we summarize our methodological contributions and draw some recommendations on how to perform a similar analysis.

11.1 Comparison with baseline studies

Table 17 compares existing literature with our average results. For each value of c, average is obtained from the prediction performance values of Table 15 and 16. Data sets that appear in both tables contribute to the average only with the best values.

Table 17: Comparison between average prediction performence of this work (*) and baselines studies.

	FP_r	TP_r	Precision
Fronza et al. (2013)	16%	65.5%	50%
Liang et al. (2007)	NA	85%	55%
Fulp et al. (2008)	NA	75%	70%
Vilalta and Ma (2002)	4%	39%	99%
Fronza et al. (2011)	27%	59.3%	18%
Salfner et al. (2006)	1%	92.3%	80%
Jiang et al. (2008)	NA	90%	93%
Shang et al. (2013)	NA	NA	33.7%
avg. $c = 1^*$	15%	80%	89%
avg. $c = 2^*$	9%	78%	95%
avg. $c = 3^*$	6%	68%	95%
avg. $c = 4^*$	4%	58%	94%

All baseline studies focus on sequences that have at least one error. Thus, in the following, we compare them with our results for $c = 1$. Just note that for $c > 1$ precision increases and FP_r and TP_r decrease. Our best result in Table 17 is obtained for $c = 2$. In terms of balance, this performance is just second to Salfner et al. (2006) who report lower precision though.

Vilalta and Ma (2002) restrict their analysis to error events of only two specific types and build features on sliding time windows. The authors propose a new rule-based model to classify sequences. Despite the high precision, performance is not fully satisfactory as the model best predicts at most 39% true positive rate over time window lengths.

Salfner et al. (2006) obtain their best result with a semi-Markov model that produces a failure probability over time based on the mean time to failure. The model is fitted on data that includes information from log messages. The model predicts a failure if probability at a given instant exceeds a time threshold. Performance and precision are very high, but they are derived from a single large set with a small proportion of defective features (Neg/Pos = 28, i.e., Pos%=3.4%). This work does not use SVMs. As we mentioned, SVMs on sets with very low percentage of positives have convergence issues, Khoshgoftaar et al. (1997). In our study, we were only able to learn linear predictors with high performance on such sets (e.g., ST_{21}, Table 18).

Comparing our findings with the work of Liang et al. (2007), Fulp et al. (2008), Jiang et al. (2008), on average and across the c values (Table 15), precision is always better. In particular, the average value of TP_r is better than the one found on a single set by Liang et al. (2007), who get a high TP_r ($\approx 85\%$) compensated with a much lower precision ($\approx 55\%$). Fulp et al. (2008), that use the same approach as in Liang et al. (2007), get a worse but still high TP_r (75%) with much better precision (70%). Still our average results are better. If we compare these three studies with ours on single data sets, Table 15 and 16, then we can find at least one case with higher precision and true positive rate for $c = 1, 2, 3$. For $c = 4$ only the work of Jiang et al. (2008) has better values, but we cannot compare it by FP_r.

Fronza et al. (2011) and Fronza et al. (2013) analyze the same data sets. As in our case, the authors build feature sets on sequences of events occurring in a time window defined by few context rules. Fronza et al. (2011) compare SVMs (L, RBF, and polynomial) with the Cox model, whereas Fronza et al. (2013) use the same SVMs but with weighted confusion matrix for training data.

Fronza et al. (2011) fit the Cox model with a set of covariates. The variables are defined by occurrences of event types in sequences. If we order them lexicographically they, in principle, correspond to our sequence type (not to our abstraction). In practice, not all the variables are used: the Cox model uses only variables that are linearly proportional to the failure hazard rate. This reduces the information entered in the model and can affect its performance for pure technical reasons.

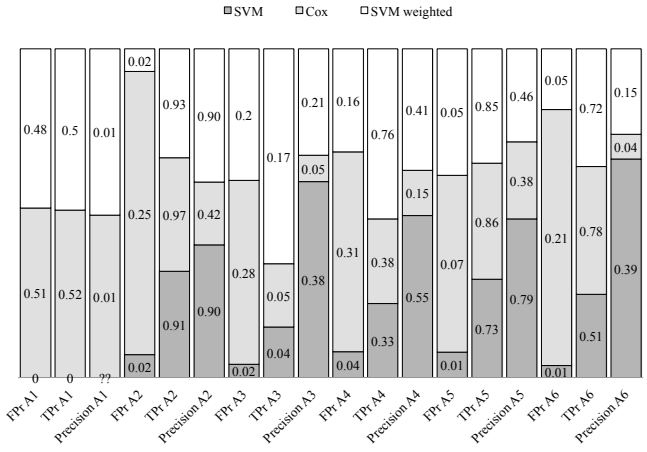
Fronza et al. (2013) use Random Indexing to distinguish sequences by event order. The algorithm encodes neighbor terms to a word. The sequence is then built incrementally. Sequence building procedure is more complex than the one in Fronza et al. (2011) and in the present study, but, in our opinion, shares the same limitations:

- The sequence abstraction does not take into account sequence frequency. On the contrary, duplicates are removed. Neglecting duplicates treats sequences that are sporadic (maybe occasional) the same way as frequent sequences (likely to be characteristic of the system). We saw that sequences with frequency greater than 1 are not rare (Fig. 2) and in a good number of features spaces, sequence frequency correlates with the number of errors (Table 7). In addition, frequent sequences that are defective might be of greater interest to system managers. We included sequence frequency in our abstraction.
- The sequence abstraction does not include the number of users that have originated the logs. We found that in a good number of feature spaces, the average number of users of a sequence correlates with

the number of errors (Table 7). We included average number of users in our abstraction.
– A sequence is built between two log-in events and limited by the duration of a day. Thus, sequences may contain events that occur after errors. This increases the noise in fitting and possibly cause overfitting (Alpaydin 2010; Hall et al. 2012).

Fig. 6 illustrates the full set of values (FP_r, TP_r, precision) for each model and feature set as described in Fronza et al. (2011) and Fronza et al. (2013).

Fig. 6: Performance and precision in Fronza et al. (2011) for SVMs and Cox model and Fronza et al. (2013) for weighted SWMs.



By visual inspection or computing balance, we can compare the different classifiers using FP_r, TP_r, since we know that the classification problem has two degree of freedom if Neg/Pos is known, Section 8.1. By computing balance, we can soon see that the weighted SVMs (RBF or L) outperform the other models for all data sets. From the figure, we can also see that the Cox Model is always less precise than both SVMs and weighted SVMs. Weighted SVMs are additionally the most precise. Thus, we decided to compare our findings with weighted SVMs, as illustrated in Table 18. Needless to say that, in this case, we are comparing different models over different features spaces and the comparison cannot lead to any conclusion in the specific context of our case study.

In Table 18, the Neg/Pos column of our six feature sets includes both the ratio on the validation sets that we used to derive precision with the conversion formula and, in brackets, the ratio of the whole feature sets. We see that the two values can substantially differ, but they are greater or less than one in a similar way. We can further see that between the two sets of feature spaces, data balance over classification groups

Table 18: Comparison between weighted SVMs in Fronza et al. (2013) (upper half) and our findings for $c = 1$, (lower half).

ID	ST	Neg/Pos	FP_r	TP_r	precision	balance
A1	12765	118.05	0.48	0.50	0.01	0.48
A2	718	5.30	0.02	0.93	0.90	0.94
A3	60	3.29	0.20	0.17	0.21	0.36
A4	343	6.79	0.16	0.76	0.41	0.79
A5	713	19.96	0.05	0.85	0.46	0.88
A6	8593	80.30	0.05	0.72	0.15	0.80
ST_2	86	1.33(1.87)	0.06	0.36	0.82	0.55
ST_3	51	0.42(0.28)	0.25	0.92	0.90	0.81
ST_6	106	1.06(1.36)	0.11	0.82	0.87	0.85
ST_{11}	220	0.19(0.31)	0.22	0.91	0.96	0.83
ST_{14}	125	0.68(0.87)	0.42	0.58	0.67	0.58
ST_{18}	305	0.18(0.16)	0.05	0.75	0.99	0.82
ST_{21}	67	32(15.75)	0.00	1.00	1.00	1.00

greatly differs: for A_1...A_6, the percentage of defective features does not reach the 25% (Neg/Pos > 3.00). The most peculiar sets are A1 and A6 for which the percentage is below the minimal threshold for convergence "4% lower bound" suggested by Khoshgoftaar et al. (1997). In these sets, precision is very low. As we mentioned in Section 8.1, this is a direct consequence of the Conversion Formula given the very high Neg/Pos ratio. Our six feature sets, cover almost the full spectrum of cases ranging between 6% (Neg/Pos =15.75) and 86% (Neg/Pos =0.16) of defective features, never reaching problematic cases. Namely, data sets with low quality of fit were filtered out or re-analyzed with balance splitting (e.g., ST_{18}). Overall, precision over our data sets is higher and performance is better

Finally, we compare our findings with prediction performance of studies on classification of faulty modules. As we mentioned, we can find many more studies in code defect prediction. We are going to use them only to draw a parallel without deriving any conclusion for our specific case study. Hall et al. (2012) have performed a deep and extensive survey on methods and results in classification of faulty modules. The best results obtained with SVMs in Hall et al. (2012) report precision $\leq 68\%$(on average 48%) and $TP_r \leq 78\%$, (on average 43%). In these regards, our results are better for $c = 1$, as good as the best results but better in precision for $c = 2, 3$, and better on average for TP_r and better in absolute for precision for $c = 4$. Hall et al. (2012) also reported that SVMs perform worse than other models. They hypothesized that "SVMs are difficult to tune and the default Weka settings are not optimal." In this study, we did not directly use Weka and we created our own scripts in R interfacing Weka via the package RWeka. In our opinion, this helped us to have a better command over parameter tuning.

11.2 What can predictions tell managers

In this section, we illustrate the case of ST_6, $c = 1$, and $t = 1/3$ to exemplify what information we can provide to managers about their system behavior.

ST_6 is the data set of an application that manages software tools of cars (e.g., keep them updated) and as such, is a pervasive application in the telemetry software system. Namely, logs of ST_6 produce 106 sequence types defined on 10 different event types, Table 7, and of which 18% with $\mu > 1$ and 89% with $\nu > 1$. With $c = 1$, we investigate the sequences that have at least one error (42%). On average, features are similarly distributed over $G_1(1)$ and $G_2(1)$ for both test and validation sets:

ST_6	Pos% test	Pos% validation	Model
t=1/2	0.4	0.45	MP
t=1/3	0.39	0.49	MP
t=1/4	0.49	0.23	MP
t=1/5	0.45	0.33	MP
avg.	**0.43**	**0.38**	**MP**

We found that MP is the best predictor for any value of the splitting parameter t and for $t = 1/3$ shows the best prediction, Table 15. After feature reduction, the number of features reduces from 12 to 7, the latter number still including both multiplicity, μ, and number of users, ν, Table 7. As such, the number of unique event types reduces from 10 to 5. In other words, there are only 5 types of events that matter for the prediction and frequent and distributed sequences are correlated with errors.

In Table 19, we derive the confusion matrix on the validation set (one-third of the original feature space) using $TP_r = 83\%$ and $FP_r = 11\%$ as found in our prediction analysis.

Table 19: Confusion matrix derived from MP predictions on ST_6, for $c = 1$, $t = 1/3$.

	Pred. Pos	Pred. Neg	Total
Pos	14 82%	3 18%	17 100%
Neg	2 11%	16 89%	18 100%
Total Percent	16 45%	19 54%	35 100%

We use the confusion matrix to predict the application future behavior. Assuming that the behavior of the application in the next three months is similar to the one studied here, suppose that 1000 sequence abstractions are being isolated. We first assume that the expected Pos percentage is the one we found on test sets, which equals to 39%. Per this percentage, we expect to have about 610 not defective and about 390 defective features. We apply MP and predict those features. Per Table 19, we would expect the model to identify about 450 sequence types as defective (45% x 1000). Developers could inspect and test these types with more accuracy. We would expect they waste time on 67 sequence types that are actually not defective (11% x 610). We would predict the model to identify correctly about 320 sequence types that are actually defective (82% x 390), but fail to identify 70 defective sequence types (18% x 390) that will cause future cost of inspection.

Based on this example, we can now read Fig. 7. The figure plots TP_r vs. FP_r on validation sets for classifiers with balance greater than 0.73 on test sets. The figure plots all the models that contribute to the average in Table 14 for any value of c. Each point in the plot corresponds to one of such models. The diagonal represents cases that give no information (i.e., the probability of a predictor to be right to fire an alert is the same as it being wrong). Few points hit the curve. Almost all models fall in the preferred area, which is the upper triangle. This area includes models that have better probability to fire a true alert than a false one. The plot also evidences the circular area of preferred balance values on prediction. As an example, the picture shows the area with balance greater than 0.73 (i.e., the radius of the circle is 0.38). In this case, the best MP models in the area appear to be more efficient than RBF in limiting the number of false positives whose presence increases inspection costs (to inspect non-defective sequence types classified as defective). We can also identify the area in which models of Table 15 lie: they correspond to points within a circle centered in (0,1) and radius 0.72. The figure also highlight the regions of higher cost for as-yet undiscovered errors and higher cost of inspection and risk-adverse as described in Moser et al. (2008) and explained in the case of ST_6. Points with low TP_r determine higher cost to fix as-yet undiscovered defective sequence types whereas high FP_r indicates higher cost due to wasted effort in inspecting sequence types that are predicted as defective, but they are not. A significant number of predictors have very low FP_r. These predictors can by of any type and take any value between 0.05 and 0.80 for TP_r.

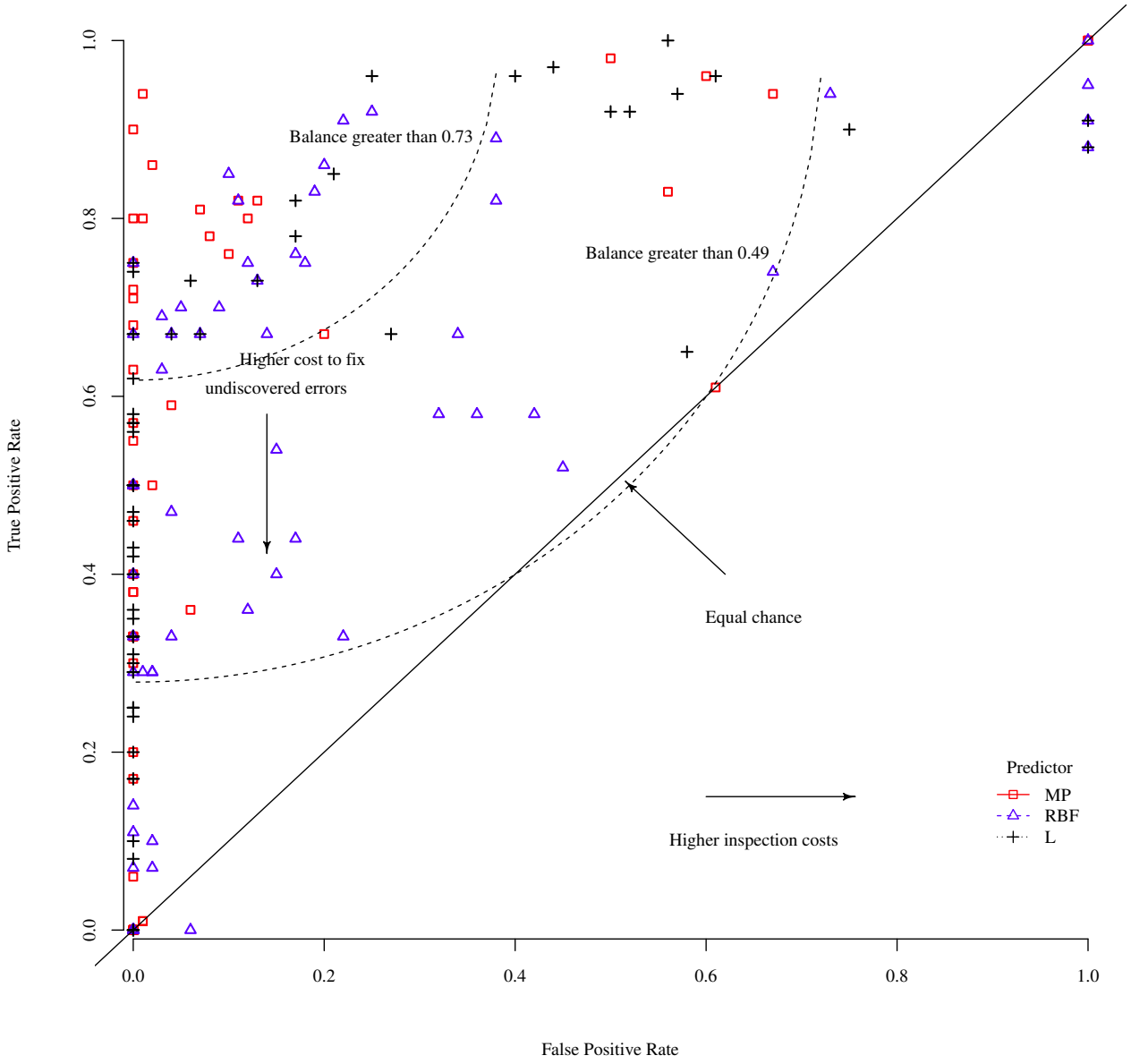


Fig. 7: Prediction performance of best models for quality of fit in case of feature reduction.

11.3 Answering Business Questions

In this section, we answer questions specific to our case study and environment. The following discussion is meant to exemplify the application of our approach and the interpretation of our results in the system we considered.

BQ1. Can we use Support Vector Machines to build suitable predictors?

Yes, in this case study, we were able to identify SVMs that effectively fit the current system behavior and predict with performance similar if not superior to existing literature, Table 17.
Specifically, we were able to find SVMs with very high precision and true positive rate both in fitting the current and predicting the future behavior. As we explained in Section 11.2, high precision and true positive rate ensure correctness of the predictions and limit the number of false negatives, i.e. risk-adverse cases when defective sequences remain as-yet undiscovered and no alarm goes off when it

should. With a low number of false negatives, SoftCar is likelier to avoid non-budgeted costs to fix unexpected system failures.
On average, SVMs also show between 4% and 17% of false positive rate. The rate decreases with the defectiveness of a sequence (e.g. $FP_r = 4\%$ for $c = 4$, Table 15). In this case, the company wastes money to inspect log sequences that are predicted defective, but, in fact, are error-free. Table 15 also indicates that when the goal is to detect sequences with a greater number of errors (i.e., $c > 1$), the waste decreases, but in parallel non-budgeted costs to fix as-yet undiscovered errors (i.e. false negatives) increases of almost the same amount.

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?

Partially, yes. In the specific case of our system, the classification problem results to be non-linear across applications. Specifically, MP is a more stable predictor and performs better than RBF for more applications and across c values, Table 15. This is also confirmed by the number of MP instances in the left top circle of Fig. 7. On the other side, RBF is more resilient after feature reduction across different splitting sizes of the same feature set. Different performance across feature spaces can be also explained by the different role that the corresponding applications have in the system. Future work will explore system behavior by sub-systems (e.g., telemetry platform, ERP system, etc.).

11.4 Answering Research Questions

RQ1. Is the amount and type of information carried by a sequence enough to predict errors?

Yes. According to our analysis on full feature spaces and in comparison with the baseline studies, our sequence abstraction allows to identify classifiers that have very good prediction performance on individual applications and good performance across applications at different cut-off values, Tables 13 and 12 respectively. The only-good performance across applications is manly due to the limited ability of SVMs to predict actual defective sequence types (highest true positive rate average = 0.41) on average. The performance is much better on individual applications, though. This confirms the findings in Hall et al. (2012) where SVMs are used as fault predictors for software modules.

RQ2. Does feature reduction increase the quality of prediction?

Yes. Although on average classifiers increase their prediction performance only a bit, on individual applications, performance increases much more and predictions tend to be independent from the splitting triplets, their size, and the cut-off values.

RQ3. Does set balancing increase the quality of prediction?

Partially, yes. Using k-splitting increases the classifier convergence for those sets on which IG does not reduce the feature space. When defective sequence types are few, balancing sets is not possible. This happens for high cut-off values as sequence types with multiple errors are rarer or occur in specific problematic applications. We also notice that with artificial balancing RBF and MP are equally represented, Table 16.

11.5 Recommendations

Reflecting on our analysis, we give here few recommendations that can guide the process to learn error predictors in the case of log sequences.

- Select classifiers with high quality of fit before any prediction analysis. No control on how models fit historical data produces predictors that do not represent the actual system behavior, but might have excellent performance on new fresh data (validation data). We have seen that no control on quality of fit can produce completely different results: in Fig 5, L is the best model on test and validation sets separately, but in Fig. 7, Tables 13, and 15 where we select only models that have high performance on test sets, MP and RBF are best predictors. Thus, we can risk to suggest managers predictions that do not reflect the actual system behavior. For this reason, we also recommend researchers to report explicitly performance of both quality of fit and generalization.
- Filter feature spaces to increase prediction ability of the classifiers. Comparing Tables 13 and 15 we see a clear increase of the precision after feature reduction. With sequences that have at least one error, predictor performance also increases while for sequences with greater number of errors ($c > 1$), true positive rate slightly decreases while false positive rate increases. This might be due to several reasons, for example, to the fact that the number of defective features decreases and more information is needed to predict them.

- Report the Pos/Neg ratio for original feature spaces, testing, and validation sets. Sets with different ratio can have different precision with the same true positive rate and false positive rate. Namely, the Conversion Formula of Zhang and Zhang (2007), Table 8, states that precision depends from true positive rate, false positive rate, and Neg/Pos ratio. As we discussed in Section 11.1, the formula also implies that models learned on spaces with a very low percentage of defective features, must have a very low false positive rate to reach high precision and true positive rate. In addition, the ratio must be reported both for test and validation sets, i.e. where performance is actually computed. This enables a correct comparison among different data sets and studies. In case the ratio is not reported, precision, true positive rate and false positive rate need to be reported.
- Control for splitting size to determine the stability of a classifier under the analysis set up. This would control data brittleness. Tables 13, 15, and 16 shows that there exist feature spaces whose best predictors change with splitting size. When possible, researchers must select predictions that do not depend on splitting size.

12 Threats to Validity

In this study, we envisage the following threats to validity according to the general framework proposed by Wohlin et al. (2012), which groups threats into four major classes: internal, construct, conclusion, and external validity threats. Some of the original threats do not apply here as this study is a retrospective analysis and does not foresee any experiment.

Internal Validity Threats: The classification is not a result of factors of which we have no control.

Neglecting multiple-threads effects in the definition of sequence types can bias the classification results. Specifically, a sequence can contain events that belong to another thread and might not be related to the event that is in "Error" state. In this case, the information in a sequence type might result augmented emphasizing the power of prediction of specific event types. Although there is no specific literature addressing this issue and given the very good precision we got across and on individual applications, we are enough confident that, in our case, even if the information were augmented it was not significantly relevant for error prediction.

Construction Validity Threats: The construction of input, output, and classifiers must reflect the classification problem.

In terms of construction validity, sequences are built under some assumptions derived by inspecting data and validated by the SoftCar unit managers. Some of the assumptions can lead to unavoidable threats, though. We identify the following ones: Neglecting time ordering in the definition of sequence abstraction can bias the classification results. Specifically, sequence with no errors can map to defective sequence types and classified as such. This implies that developers can waste more time in inspecting defective sequences that the time predicted by our models (e.g., MP). Such circumstances can only happen when defective sequence types have multiplicity μ greater than one. On average, for the 25 application of our study, the number of such sequence types is less than 8% which is within the standard acceptable tolerance threshold, but in any case must be considered in the final cost computation.

Conclusion Validity Threats: The analysis design must ensure statistical significance of the proposed solution.

We controlled for different threats in this class. We provided a detailed descriptive analysis to allow drawing conclusions. We used cross-validation and repetition of the analysis across splitting triplets to avoid the influence of analysis set up. We compare statistical techniques to select the more powerful on the case study data and control for data brittleness. We compared with baseline studies the accuracy of our predictions and the method used in terms of sequence abstraction, model used, analysis set up, and results. We repeated our analysis on 25 different data sets. Yet, we did not repeat the analysis with different types of classifiers (i.e., Bayesian models) that seem promising with software modules (Menzies et al. 2007b; Hall et al. 2012).

External Validity Threats: the analysis must ensure generalizability.

In term of external validity, this study, as any other empirical single-case study, is subjected to the limitation of the specific environment. Even though we replicated the analysis over 100 original, heterogeneous data sets, these sets originated from the same system. To our knowledge there exist only few studies in execution logs that classify errors on different systems or system versions, e.g., (Shang et al. 2013). Literature on system log analysis does not report any of these studies, Table 2.

13 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.

14 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

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. *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.

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.

G. Denaro and M. Pezzè. An empirical evaluation of fault-proneness 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.

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, 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. 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.

I. Fronza, A. Sillitti, G. Succi, T. Mikko, and Vlasenko J. Failure prediction based on log files using random indexing and support vector machines. *Journal of Systems and Software*, 86(1):2 – 11, 2013.

S. Fu and C. Z. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, SC '07, pages 41:1–41:12, New York, NY, USA, 2007. ACM.

S. Fu and C. Z. Xu. Quantifying event correlations for proactive failure management in networked computing systems. *J. Parallel Distrib. Comput.*, 70(11):1100–1109, November 2010. ISSN 0743-7315.

E. W. Fulp, G. A. Fink, and J. N. Haack. Predicting computer system failures using support vector machines. In *Proceedings of the First USENIX conference on Analysis of system logs*, WASL'08, pages 5–5, Berkeley, CA, USA, 2008. USENIX Association.

D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the nasa metrics data program data sets for automated software defect prediction. In *Evaluation Assessment in Software Engineering (EASE 2011), 15th Annual Conference on*, pages 96 –103, april 2011.

K. C. Gross, V. Bhardwaj, and R. Bickford. Proactive detection of software aging mechanisms in performance critical computers. In *Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE*, pages 17 – 23, dec. 2002.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, 2012.

A. W. Jayawardena, D. A. K. Fernando, and M. C. Zhou. Comparison of multilayer perceptron and radial basis function networks as tools for flood forecasting. *IAHS Publications-Series of Proceedings and Reports-Intern Assoc Hydrological Sciences*, 239:173–182, 1997.

W. Jiang, C. Hu, S. Pasupathy, A. Kanevsky, Z. Li, and Y. Zhou. Understanding customer problem troubleshooting from storage system logs. In *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, pages 43–56, Berkeley, CA, USA, 2009a. USENIX Association.

Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. An automated approach for abstracting execution logs to execution events. *J. Softw. Maint. Evol.*, 20(4):249–267, July 2008. ISSN 1532-060X.

Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Automated performance analysis of load tests. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 125–134, 2009b.

J. Kent. Information gain and a general measure of correlation. *Biometrika*, 70(1):163–173, 1983.

T. M. Khoshgoftaar and D. L. Lanning. A neural network approach for early detection of program modules having high risk in the maintenance phase. In *Selected papers of the sixth annual Oregon workshop on Software metrics*, pages 85–91, New York, NY, USA, 1995. Elsevier Science Inc.

T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, and S. J. Aud. Application of neural networks to software quality modeling of a very large telecommunications system. *Trans. Neur. Netw.*, 8(4):902–909, jul 1997. ISSN 1045-9227.

T. M. Khoshgoftaar, Kehan Gao, and N. Seliya. Attribute selection and imbalanced data: Problems in software defect prediction. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, volume 1, pages 137 –144, oct. 2010.

D. Kim, X. Wang, S. Kim, A. Zeller, S. C. Cheung, and S. Park. Which crashes should i fix first?: Predicting top crashes at an early stage to prioritize debugging efforts. *IEEE Trans. Softw. Eng.*, 37(3):430–447, May 2011. ISSN 0098-5589.

G. Le Gall, M.-F. Adam, H. Derriennic, B. Moreau, and N. Valette. Studies on measuring software. *Selected Areas in Communications, IEEE Journal on*, 8(2):234 –246, feb 1990. ISSN 0733-8716.

C. S. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Pacific Symposium on Biocomputing*, pages 566–575, 2002.

Z. Li, S. Zhou, S. Choubey, and C. Sievenpiper. Failure event prediction using the cox proportional hazard model driven by frequent failure signatures. *IIE Transactions*, 39(3): 303–315, 2007.

Y. Liang, Y. Zhang, X. Hui, and R. Sahoo. Failure prediction in ibm bluegene/l event logs. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 583–588, 2007.

Y. Liu, T. M. Khoshgoftaar, and N. Seliya. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Trans. Softw. Eng.*, 36(6):852–864, November 2010. ISSN 0098-5589.

D. Lo, H. Cheng, J. Han, S. C. Khoo, and C. Sun. Classification of software behaviors for failure detection: a discriminative pattern mining approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 557–566, New York, NY, USA, 2009. ACM.

H. Mannila, H. Toivonen, and V. A. Inkeri. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, January 1997. ISSN 1384-5810.

T. Menzies, A. Dekhtyar, J. S. Di Stefano, and J. Greenwald. Problems with precision: A response to "comments on 'data mining static code attributes to learn defect predictors'". *IEEE Trans. Software Eng.*, 33(9):637–640, 2007a.

T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Trans. Softw. Eng.*, 33(1):2–13, January 2007b. ISSN 0098-5589.

R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 181–190, New York, NY, USA, 2008. ACM.

J.C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *Software Engineering, IEEE Transactions*

on, 18(5):423–433, 1992.

N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 284–292, New York, NY, USA, 2005. ACM.

N. Nagappan, B. Murphy, and V. Basili. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 521–530, New York, NY, USA, 2008. ACM.

N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 309–318, 2010.

A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP International Conference on*, pages 575–584, 2007.

A. Oliner, A. Ganapathi, and W. Xu. Advances and challenges in log analysis. *Commun. ACM*, 55(2):55–61, February 2012. ISSN 0001-0782.

A. J. Oliner, A. Aiken, and J. Stearley. Alert detection in system logs. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 959–964, 2008.

E. Pinheiro, W. D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, FAST '07, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1267903.1267905`.

A. A. Porter and R. W. Selby. Empirically guided software development using metric-based classification trees. *IEEE Softw.*, 7(2):46–54, March 1990. ISSN 0740-7459.

J. R. Quinlan. Simplifying decision trees. *International Journal of ManMachine Studies*, 27:221–234, 1987.

B. Russo. Parametric classication over multiple samples. In *DAPSE13: International Workshop on Data Analysis Patterns in Software Engineering, Proceedings of, collocated with ICSE 2013, San Francisco, California, USA, May 21, 2013*, 2013.

M. Sahlgren. An introduction to random indexing. In *In Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*, 2005.

F. Salfner, M. Schieschke, and M. Malek. Predicting failures of computer systems: a case study for a telecommunication system. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8 pp.–, 2006.

W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin. Assisting developers of big data analytics applications when deploying on hadoop clouds. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 402–411, Piscataway, NJ, USA, 2013. IEEE Press.

M. Steinder and A. S. Sethi. Probabilistic fault localization in communication systems using belief networks. *IEEE/ACM Trans. Netw.*, 12(5):809–822, October 2004. ISSN 1063-6692.

R. Valette, J. Cardoso, and D. Dubois. Monitoring manufacturing systems by means of petri nets with imprecise markings. In *Proceedings of the IEEE International Symposium on Intelligent Control, 1989, Albany, NY, USA*, pages 233–238, Washington, DC, USA, 1989. IEEE Comput. Soc. Press.

W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, November 2003. ISSN 0169-023X.

V. N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

R. Vilalta and S. Ma. Predicting rare events in temporal domains. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 474–, Washington, DC, USA, 2002. IEEE Computer Society.

C. Wohlin, P. Runeson, M. Hoest, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, 2012.

Xi. Wu, V. Kumar, J. Ross Q., J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, December 2007. ISSN 0219-1377.

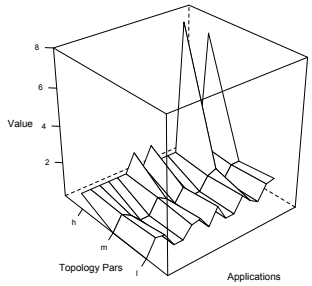
F. Xing, P. Guo, and M. R. Lyu. A novel method for early software quality prediction based on support vector machine. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, ISSRE '05, pages 213–222, Washington, DC, USA, 2005. IEEE Computer Society.

K. Yamanishi and Y. Maruyama. Dynamic syslog mining for network failure monitoring. In *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 499–508, New York, NY, USA, 2005. ACM.

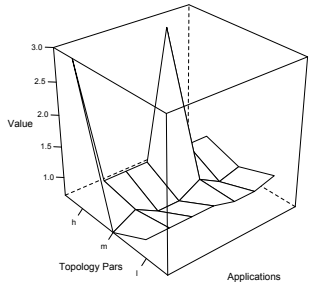
H. Zhang and X. Zhang. Comments on "data mining static code attributes to learn defect predictors". *Software Engineering, IEEE Transactions on*, 33(9):635–637, 2007.

Appendix

Figure 8 shows the MP topology optimal values (z-axis) across the 25 data sets for one specific choice of the cut-off value and splitting parameter.



(a) c=2, t=1/3



(b) c=2, k=2

Fig. 8: MP topologies (parameters k, l, and m) over applications. Cut-off $c = 2$. Topology values are obtained on the training and test sets of a splitting triplet. Left: t-splitting with $t = 1/3$. Right: k-splitting with number of zero defects $k = 2$.