Log-based Predictive Maintenance

Ruben Sipos[*]
Dept. of Computer Science
Cornell University
Ithaca, NY
rs@cs.cornell.edu

Dmitriy Fradkin
Corporate Technology
Siemens Corporation
Princeton, NJ
dmitriy.fradkin@siemens.com

Fabian Moerchen[*]
Amazon
Seattle, WA
fabian@mybytes.de

Zhuang Wang[*]
Skytree
San Jose, CA
john.wang@skytree.net

ABSTRACT

Predictive maintenance strives to anticipate equipment failures to allow for advance scheduling of corrective maintenance, thereby preventing unexpected equipment downtime and improving service quality for the customers. We present a data-driven approach based on multiple-instance learning for predicting equipment failures by mining equipment event logs which, while usually not designed for predicting failures, contain rich operational information. We discuss problem domain and formulation, evaluation metrics and predictive maintenance work flow. We experimentally compare our approach to competing methods. For evaluation, we use real life datasets with billions of log messages from two large fleets of medical equipment. We share insights gained from mining such data. Our predictive maintenance approach, deployed by a major medical device provider over the past several months, learns and evaluates predictive models from terabytes of log data, and actively monitors thousands of medical scanners.

1. INTRODUCTION

Success of manufacturing companies largely depends on reliability of their products. Scheduled maintenance is widely used to ensure that equipment is operating correctly so as to avoid unexpected breakdowns. Such maintenance is often carried out separately for every component, based on its usage or simply on some fixed schedule. However, scheduled maintenance is labor-intensive and ineffective in identifying problems that develop between technician's visits. Unforeseen failures still frequently occur. In contrast, *predictive maintenance* techniques help determine the condition of in-service equipment in order to predict when and what repairs should be performed. The main goal of predictive maintenance is to enable pro-active scheduling of corrective work, and thus prevent unexpected equipment failures.

Predictive maintenance requires insight into the running condition of equipment. This can be gained by adding sensors to the equipment for recording and monitoring of signals of interest (such as temperature and voltage). The predictive maintenance module can then send alerts when sensor values deviate from normal ranges. Though sometimes an effective solution, it is impractical for in-service equipment, since major hardware upgrades, such as adding sensors, are often infeasible, especially on large fleets, due to cost, effort and potential regulatory hurdles. Alternatively, one can gain insight into the workings of a piece of equipment by studying its logs. Modern equipment is usually operated via software applications. For example, in case of medical scanners, all device operations, from warming up to scanning a patient and from generating a medical report to calibration, are controlled by various applications. These applications produce logs of their operation. These logs reflect the developers' original ideas about what are the valuable events to report, and contain informational or error messages, internal states, or exceptions. Theoretically, one can trace back how a piece of equipment was used by analyzing its logs. Mining such rich information can help in detecting potential issues in advance.

The use of equipment logs to predict failures poses many challenges and has not yet been fully explored. Since logs are mainly used for debugging purposes, they (i) rarely contain explicit information for failure prediction; (ii) contain heterogeneous data including symbolic sequence, numeric time series, categorical variables and unstructured text; and (iii) contain massive amounts of data, posing computational challenges. To make use of log data, we first have to interpret the logs, filter out a large amount of noise (i.e. data irrelevant to our goal) and extract predictive features. Next, we have to collect known failure cases for learning/evaluating models, transform the problem into an appropriate learning scenario and determine a performance measure that reflects real-world needs. Then, we have to apply machine learning techniques to effectively and efficiently solve the learning problem. Moreover, we have to take into account specifics of the domain. In the following sections we discuss these points in more detail with the focus on the data and do-

[*] This work was done while at Siemens Corporation.

KDD'14, August 24–27, 2014, New York, NY, USA.
Copyright 2014 ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623340.

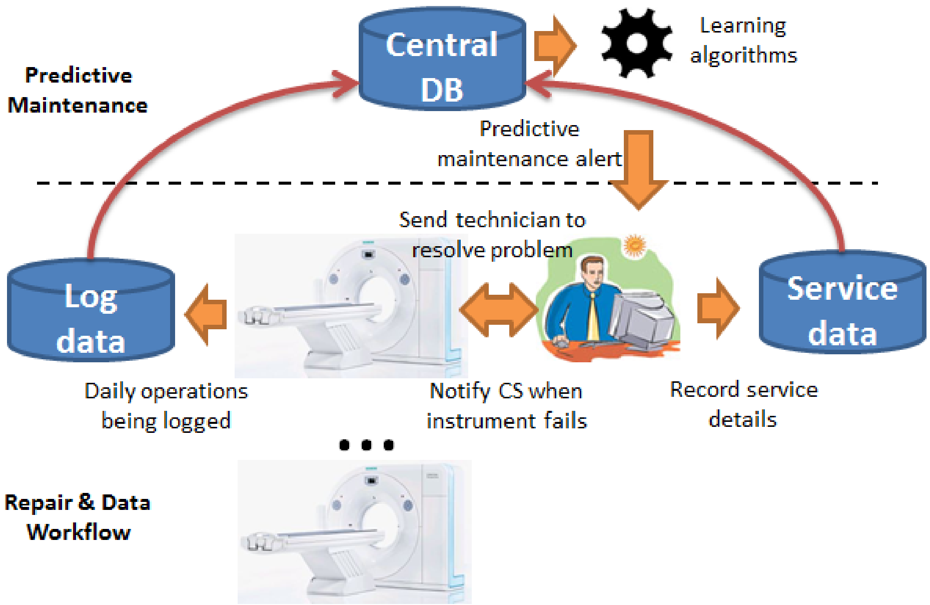

Predictive Maintenance
Central DB
Learning algorithms
Predictive maintenance alert
Send technician to resolve problem
Log data
Service data
Daily operations being logged
Notify CS when instrument fails
Record service details
• • •
Repair & Data Workflow

Figure 1: Predictive maintenance workflow.

main description, the problem formulation and a case study, which serve as our main contributions.

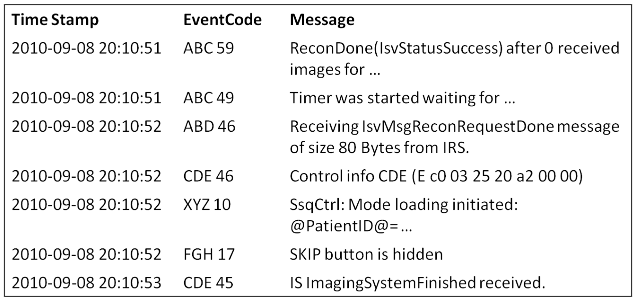
2. PROBLEM DESCRIPTION

In this section, we describe our data and application.

2.1 Data Description

A typical service life cycle looks as follows: equipment operates normally at first. When a problem occurs, the customer calls the service center for support, a "notification" is opened and repair service is scheduled. Then a technician comes on-site to resolve the problem. After resolution, the technician updates the notification with repair details such as component consumption information, hours spent, trouble-shooting or repair description, and closes the notification. Throughout the whole process, during both normal and abnormal operation, the equipment automatically records all events into log files. This cycle (illustrated in the bottom part of Figure 1) gets repeated over and over in a fleet of thousands of equipment units. The log files from such a fleet collected over the past several years form our data in this study. [1]

Log data is a collection of **events** recorded by various applications running on the equipment (see Figure 2). An event consists of a timestamp (indicating when the event occurred), a message text (either fully unstructured or generated from a template) describing the event, an event code (representing the category of a group of similar message text) and event severity. Events reflect the developers' original idea about what are the valuable states to report. In our application, we have thousands of unique event codes and a theoretically unlimited number of distinct messages texts.



Time Stamp	EventCode	Message
2010-09-08 20:10:51	ABC 59	ReconDone(IsvStatusSuccess) after 0 received images for ...
2010-09-08 20:10:51	ABC 49	Timer was started waiting for ...
2010-09-08 20:10:52	ABD 46	Receiving IsvMsgReconRequestDone message of size 80 Bytes from IRS.
2010-09-08 20:10:52	CDE 46	Control info CDE (E c0 03 25 20 a2 00 00)
2010-09-08 20:10:52	XYZ 10	SsqCtrl: Mode loading initiated: @PatientID@= ...
2010-09-08 20:10:52	FGH 17	SKIP button is hidden
2010-09-08 20:10:53	CDE 45	IS ImagingSystemFinished received.

Figure 2: A piece of a real log file (with renamed event codes and messages) from a medical scanner.

An equipment log is usually broken into day-long pieces. A daily log can contain tens of thousands of events since their time resolution can be as small as a few seconds.

Log data is unique in several aspects. It is temporal, and can be viewed both as symbolic sequences (over event codes) and as numeric time series, with variables extracted from messages or with event frequencies over some window, e.g. days. It can additionally include categorical features (event codes, and categorical variables in text) and fully unstructured data such as message text (some text similarity between message text may make sense, in particular in the absence of event code categories).

An experienced domain expert can scan equipment logs to identify abnormal conditions. The old-fashioned predictive maintenance approach in this domain is to manually create predictive patterns for a particular component based on the boolean combination of a few relevant event codes. Such an approach is heavily experience-based and very time consuming, but it illustrates an important concept - that component failure can be predicted by checking daily logs for patterns

[1] Although we use the data from medical devices to demonstrate our approach, our approach is also applicable to other domains, such as IT infrastructure or industrial equipment in which equipment logs are recorded.

consisting of multiple event codes. This concept motivates us and serves as the basis for our problem formulation.

Service data is another crucial data type in predictive maintenance applications. Service notifications (or tickets) are used by the service center to record details of performed services such as the notification open date (i.e. the date customer reports a problem), the equipment involved, the component consumption and so on. Under a reasonable, though not always correct, assumption that component replacement is the consequence of a failure of that particular component, we can use this information to correlate a known component failure with its corresponding equipment, time and relevant logs.

However, service data is noisy and often includes incorrect information. A replacement of a component might be ordered but not used and later sent back to the factory; or a component may be exchanged but the repair may not have been appropriate for the problem at hand. Also, some failure modes do not involve component exchanges and need to be identified by matching certain keywords in the notification text such as "greasing" or "recalibrating". We formulate the problem using a robust learning approach (described in Section 3), which also serves to reduce the impact of low-quality data.

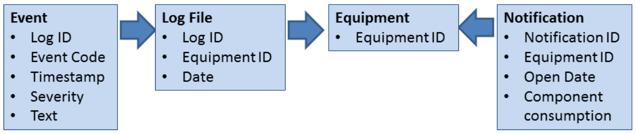


Figure 3: An ER diagram for predictive maintenance.

Figure 3 shows logical relationships between entities in predictive maintenance domains.

Given a target component and a collection of historical equipment log and service data, we can formulate a predictive maintenance problem for the target component. It consists of constructing a binary classifier for predicting failures, where training examples contain predictive features extracted from an interval of equipment's log, and the label is determined by the occurrence of the component replacement in the service data after that interval. Note that our approach is not specific to certain failure types, but targets all hardware failures (within business needs) that can be rectified by replacing or repairing the failing component and can be detected via logged data caused by aberrant or inoperable behavior. We give a more formal problem formulation in Section 3.

2.2 Predictive Maintenance Workflow

The predictive maintenance workflow is illustrated in the top part of Figure 1. A central database is at the core of a predictive maintenance platform. Data from in-service equipment and the Support Centers is collected and integrated there. The analytic module processes data from the central database. Analysis consists of the following stages: data preparation, model building, model evaluation, and monitoring. To build a model for a target component, the analytic module first pulls the relevant data from the central database, extracts predictive features (see Section 3), transforms and represents the data in a matrix format for learning algorithms. Learning algorithms build models and then pass them on for evaluation. A model is evaluated against historical data and scored by several key performance indicators. After evaluation, domain experts review the model's performance and decide whether to deploy the model into the system. In the monitoring module, the system pulls the new daily log data from monitored equipments and predict a component failure using a model. If predicted score exceeds a predefined threshold, an alert will be sent to the service center for further review.

2.3 Requirements

We worked closely with domain experts to come up with requirements that would make a log-based predictive maintenance solution practical and useful. These requirements, summarized below, served as a guideline for our design.

The timing of an alert is important for the evaluation of the quality of a model in predictive maintenance:

- **Predictive Interval**: a pre-defined time interval right before a failure. An alert occurring in this interval gives enough time for the support center to act and is considered successful.
- **Infected Interval**: a pre-defined time interval right after a failure. The equipment is breaking down or under repair. Data from this interval should not be used for modeling/evaluation.
- **Responsive Duration**: a pre-defined time length reflecting real-life action time for an alert.

True and false positives are then defined as:

- **True Positive**: an alert that occurs in the predictive interval. Note that multiple alerts in the same predictive interval only count as one true positive.
- **False Positive**: an alert that occurs outside the predictive/infected interval. Note that multiple such alerts within a responsive duration only count as one false positive.

Finally, using the definitions above, we can evaluate the performance of a model using precision and recall:

- **Precision**: True Positive / (True Positive + False Positive).
- **Recall**: True Positive / All failures.

The maintenance strategy is determined separately for different components and is influenced by many factors such as the repair cost, failure severity and the business model. Learning algorithms should balance Precision and Recall and provide user flexibility in trading these off. Without knowledge of the specific maintenance strategy, we can evaluate a model with Predictive-Maintenance-based Area Under precision-recall Curve (**PM-AUC**) score as a simple measurement. PM-AUC is computed like regular AUC but using the above definitions of Recall and Precision.

Other requirements are not defined as formally, but are present nonetheless, so we discuss them briefly.

Interpretability: It is preferable to have an interpretable model so that experts are able to review it. This allows for incorporation of expert feedback into the modeling process. We use methods with L1 regularization to build sparse linear model which the domain experts can easily review. Such a model consists of a weighted sum of a relatively small numbers of predictive features, with the weights specifying their precise contributions to the decision. Furthermore, known noisy or irrelevant features specified by experts can be easily excluded from the new modeling process. In some cases just knowing that the model is "interpretable" and understanding which features affect model performance makes experts more comfortable in adopting it.

Efficiency: The learning algorithm should be fast and capable of handling hundreds of thousands of training examples in a space of tens of thousands features. We use the state-of-the-art sparse linear classifier package Liblinear [9] to achieve this.

Handling Class Imbalance: The number of known failures is usually small. We have to be able to learn models with only tens of known failure cases. To deal with such extremely imbalanced labeled data, we apply a combination of stratified sampling, stable feature selection and large margin techniques to prevent over-fitting and learn a robust model.

3. METHODOLOGY

We view our problem as an example of Multi-Instance Learning (MIL) [8] because it is a good fit for our setting. In MIL, instead of receiving a set of independent labeled instances as in standard classification, the learner receives a set of bags which are labeled positive or negative. Each bag may contain multiple instances (equipment daily logs). A bag (all the logs from an interval, e.g. one week) is labeled negative if all the instances in it are negative (no failures according to the service notifications), and positive if it contains at least one positive. Given bags obtained from different equipment and at different dates, the goal is to build a classifier that will label either unseen bags or unseen instances correctly. The accuracy of models is here measured at the bag level.

MIL formulation captures several important aspects of our problem:

- It is more realistic to assume that at least one daily log within a short interval before a failure carries a failure signature than to assume that all the daily logs within a short interval before a failure carry a failure signature. More likely, equipment continues working irregularly, switching between normal and abnormal conditions, before a final breakdown. Therefore we should not label all instances in a pre-failure interval as positive (as we would in a non-MIL setting).

- All the daily logs are normal within a non-failure interval

- The goal is to correctly label a bag rather than every instance. This is strictly consistent with the domain-based evaluation metric.

We formally define the learning task as follows. Let D be a set of B labeled bags, $D = \{bag_j, j = 1, ..., B\}$, where $bag_j = (\{x_{ij}, i = 1, ..., b_j\}, y_j)$, $x_{ij} \in R^d$ is a feature vector of the i-th instance from the j-th bag, $y_j \in \{+1, -1\}$ is the binary label of the j-th bag and b_j is the number of instances in the j-th bag. The objective[2] is to learn a model f whose decision function $sgn(f(bag))$ accurately predicts the label of a bag.

One of the major challenges in our problem domain comes from the label imbalance and the low quality of positive labels. Since labeling is based on service notifications which are not entirely reliable, even slightly inaccurate notification dates would significantly change the identification of positive bags, if we did not take this into account. Moreover, as the known failure cases are very rare, any noisy positive bags would significantly downgrade the model quality. Recent advances in MIL resulted in many successful algorithms for learning f, but most of them are not robust to label noise in the situations with imbalanced data. Below we describe a simple algorithm which is robust to label noise for the rare class and is a more effective solution than alternatives, as evidenced by our experimental evaluation described in Section 4.

Training. Many different ways of solving MIL exist[10]; our approach combines instance-to-example, meta-examples and instance-based supervised learning.

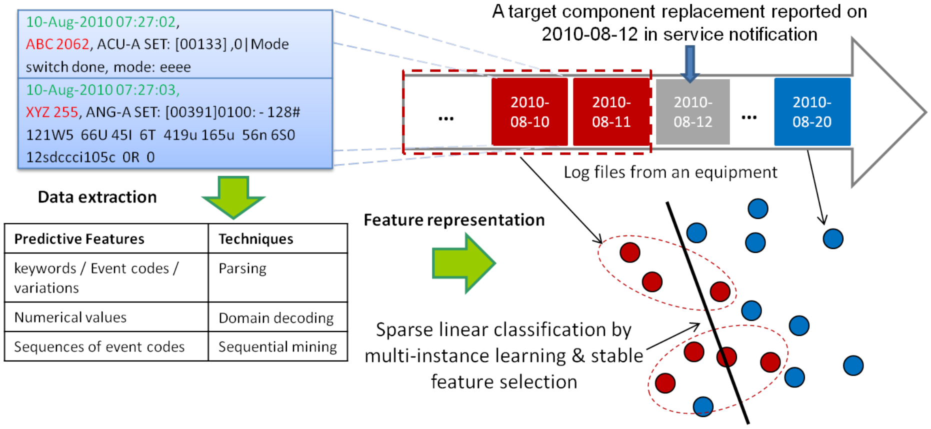
We first transform the MIL dataset as follows: if the j-th bag is negative, we create multiple negative examples $(x_{ij}, -1)$ by an instance-to-example approach (i.e. each instance in a negative bag becomes one negative example), because all instances in negative bags are part of normal operation and not indicative of failure. If the j-th bag is positive, we create s positive example by averaging all its instances into a single positive meta example $(x_j, +1)$, where $x_j = mean(\{x_{ij}\}, i = 1, ..., b_j)$ (i.e. each positive bag becomes one positive example). One rationale for this is that the new positive meta-example (i.e. average over all instances in a positive bag) is guaranteed to be positive since we know there is at least one positive instance in the bag. Although it compromises some level of the discriminative power of positive examples due to inclusion of features from negative instances, the averaging strategy improves the label quality of positive examples which is more critical for imbalanced data.

Joining all the newly created examples from all the bags into a single training dataset $D' = \{(x_j, y_j), j = 1, ..., M\}$, we formulate the learning problem as a L1-regularized SVM optimization, $\min_w \frac{\lambda}{2}||w||_1^2 + \sum_j \max\{1 - y_j w^T x_j, 0\}$, where $\lambda > 0$ is a user-specified regularization parameter. The optimization can be efficiently solved by Liblinear [9].

Prediction. In MIL, how to apply the learned model on new instances / bags depends on the application requirement. It is not uncommon to apply different prediction strategy in training/testing phase. In our real-life deployment phase, new log data (i.e. instance) arrives on a daily base. We apply the learned model to individual instances as they arrive and trigger an alert if the prediction is beyond a pre-defined threshold (which is selected during the testing phase based on a desired precision/recall point on the PM-AUC curve). To be aligned with this setup, in the testing phase, we do not transform bags. Instead, we predict every instance of a bag and predict the bag as positive if the prediction score of a bag instance hits a given threshold[3], otherwise as negative. In the case of true pos-

[2] Actually, this 0-1-error-based objective is a surrogate as directly maximizing PM-AUC is difficult.

[3] We increment the threshold and repeat the process to calculate PM-AUC.



```
10-Aug-2010 07:27:02,
ABC 2062, ACU-A SET: [00133] ,0|Mode
switch done, mode: eeee
10-Aug-2010 07:27:03,
XYZ 255, ANG-A SET: [00391]0100: - 128#
121W5  66U 45I  6T  419u 165u  56n 6S0
12sdccci105c  0R  0
```

Predictive Features	Techniques
keywords / Event codes / variations	Parsing
Numerical values	Domain decoding
Sequences of event codes	Sequential mining

A target component replacement reported on
2010-08-12 in service notification
...
2010-08-10
2010-08-11
2010-08-12
...
2010-08-20
Log files from an equipment
Data extraction
Feature representation
Sparse linear classification by
multi-instance learning & stable
feature selection

Figure 4: Our methodology.

itive prediction, our model gives an alert between one[4] to length-of-predictive-interval days in advance.

Feature Representation We use a bag-of-features approach to create feature vectors. The features that can be extracted from logs include:

- Keywords: bag-of-keyword is a fine-grained feature representation which can easily blow up the feature space to millions of dimensions. Since it poses computational challenges and is usually difficult to interpret in the final model, we do not use keywords in this paper.

- Numerical values: Numerical values are often used to record important physical or mechanical conditions. Extracting such values from text is a highly domain-dependent task because the values are often encoded to hide sensitive information. Given the high cost of domain effort for decoding, we only extract a limited number of numerical values and represent each with several statistic-based features.

- Event codes: an event code is the developer-specified category of a group of similar message texts. It is a high-level summary of the event content. Since event codes are domain-friendly (as they are widely used in traditional pattern creation process) and the extraction is fully domain-independent, we use them as basic features.

- Event code variations: textually similar messages from the same event code might be logically different. For example, "the scanning button is on" and "the scanning button is off" come from the same event code but record two opposite operations. To distinguish semantically different messages from the same event code, we decompose event codes into event code variations. For all the message text from the same event code, we first filter out embedded numerical values and then identify each distinct message template as a new event code variation. This type of feature is highly desirable by domain experts.

- Sequences: sequential features capture the logical order of events and enhance the interpretability of the final model. We have also conducted initial experiments with event-code-based sequence features generated by a novel sequential pattern-mining algorithm designed for MIL. We do not discuss details here due to space constraints and leave it for future work.

Feature Selection We use a bootstrapped feature selection algorithm to select a subset of relevant features for building robust learning models with imbalanced labels. In the spirit of [4], we perform stable feature selection by training multiple sparse linear classifiers under stratified subsampling. We create each sample of data by including all the positive bags and a set of randomly picked negative bags. We train a sparse linear classifier on the sample and learn the model weights w_i, where i is the index of a sample of data. After repeating this process multiple times, we calculate $|\sum_i w_i|$, rank features by these values and then select the features by retaining only those with the highest rankings. Afterwards, the selected features are used for another sparse linear classification on all the training data to learn the final model.

Our methodology is summarized on Figure 4. Creating the dataset, training of the model and how we test the model are described in Algorithm 1.

4. EXPERIMENTS

Our data has been collected over the several years from two large fleets of medical equipments from a major medical device provider. For each type of equipment we choose a target component of high practical interest and with a sufficient number of known failure cases. We use 7 days for the Predictive Interval, 20 days for the Infected Internal and 7 days for the Responsive Duration for both predictive maintenance tasks. All these values were agreed upon with domain experts (and we did not find significant differences in performance by slightly changing bag sizes). For each data set we create positive bags from the the daily logs of [-7 day, -1 day] interval before each failure, where day 0 is the notification open date (i.e. the noted day of the failure), and negative bags by randomly selecting about 20% of all the

[4] In the worst case the alert would be raised based on the last daily log before the failure.

Algorithm 1

```
Build the MIL dataset:
  1. Parse daily event logs to extract features.
  2. Parse service notifications to extract known failure information.
  3. Create bags for MIL learning.
     (a) Group daily instances into bags using desired interval lengths.
     (b) Label bags with known failures as positive and the rest as negative.
Train the model:
  1. In the training phase, transform the MIL dataset.
     (a) Each instance in a negative bag is transformed into a negative example.
     (b) For each positive bag we create a positive meta example using the average of all bag's instances.
  2. Feature selection.
     (a) Create multiple subsets by randomly subsampling transformed negative bags and including all positive meta examples.
     (b) Learn a sparse linear classifier on each subset.
     (c) Average weights from all runs and select features with the highest absolute weights.
  3. Train the final model.
     (a) Use the subset of features obtained in the previous step.
     (b) Learn the final model by using all the data.
Test/Apply the model:
  1. In evaluation, predict a bag as positive if the prediction score of a bag instance hits a given threshold.
  2. In deployment, trigger an alert if the prediction score of a new daily log hits the pre-defined threshold.
```

C	A	description
16389	75552	#instances
3073	11238	#features
88	108	#known failures
6664	14367	#bags

Figure 5: Dataset summary.

	C	A
random	0.037 (0.004)	0.017 (0.003)
AllInst.	0.293 (0.014)	0.620 (0.013)
Agg.	0.174 (0.013)	0.498 (0.016)
MILES	0.170 (0.011)	0.427 (0.117)
MI-SVM	0.216 (0.038)	0.700 (0.014)
Ours	**0.319 (0.015)**	**0.730 (0.011)**

Figure 6: PM-AUC comparison.

remaining weekly intervals. The resulting datasets A and C are summarized in Figure 5.

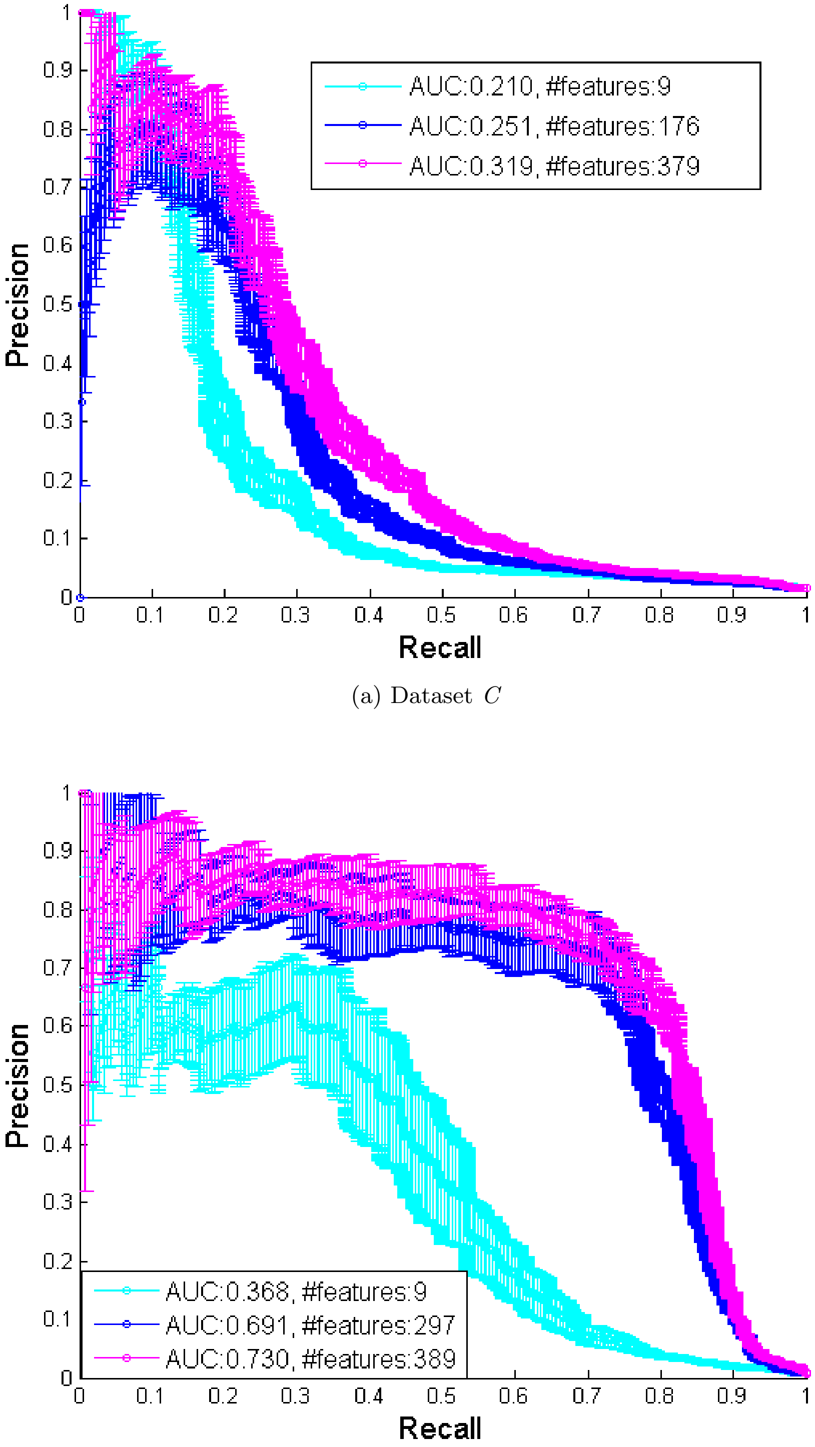
Domain-based Evaluation We compare our algorithm against the following methods:

- **AllInstances:** A baseline wrapper algorithm that transforms MIL into standard supervised learning by assigning the bag label to its instances (i.e. a pure *instance-to-example* way of solving MIL). It was reported to work well in some cases [18].
- **Aggregated**[23]: Another baseline wrapper algorithm that aggregates each bag into a meta-example by averaging (i.e. a pure *meta-example* way of solving MIL).
- **MILES**[6]: A popular MIL wrapper algorithm that embeds each bag into a new feature space defined by distances to training instances.
- **MI-SVM**[2]: An iterative algorithm that solves the optimization formulation of MIL as a maximum margin problem.

The PM-AUC scores of these methods are shown in Table 6. For the wrapper algorithms, L1-regularized Linear SVM with the bootstrapped feature selection is used for supervised learning[5]. Experimental results (mean and standard deviation) are reported based on bag-level 5-fold cross validation with stratified sampling for imbalanced positive labels. All hyper-parameters were selected using cross validation. These results show that our approach significantly outperforms other methods in terms of PM-AUC.

The maintenance strategy is determined by business units separately for different equipment or component, which is why predictive models have to allow trading-off between precision and recall according to the specific strategy. Figure 7 contains the PM-ROC curves (showing the mean and 90% confidence interval from 5-fold cross validation) of the models with different complexity (controlled by adjusting the L-1 regularization parameter) learned by our algorithm. In both figures, we can observe that the pink curves, representing models composed of 300 to 400 predictive features, achieve the best performance. If the business requirements are for 70% precision then our best models can cover 25% and 80%

[5] We also explored kernel SVM but did not observe significant performance improvement.



(a) Dataset C

(b) Dataset A

Figure 7: PM-ROC plots with different model complexity learned by our algorithm.

of failures (i.e. recall at that precision point) within the 7-day Predictive Interval for each dataset, respectively. Considering that no existing predictive rules/models are available for these two components, our method is highly successful. Simpler models, with fewer features, are also desirable from both the machine learning and the domain points of view. Though the most complicated models (with 300 to 400 features) achieve the best PM-AUC scores on both datasets, the simpler models with less than ten features and suboptimal scores are still useful. Their simplicity, together with relatively high performance (high precision and reasonable recall), enable domain experts to understand them and thus help detect root causes of failures or identify specific failure modes.

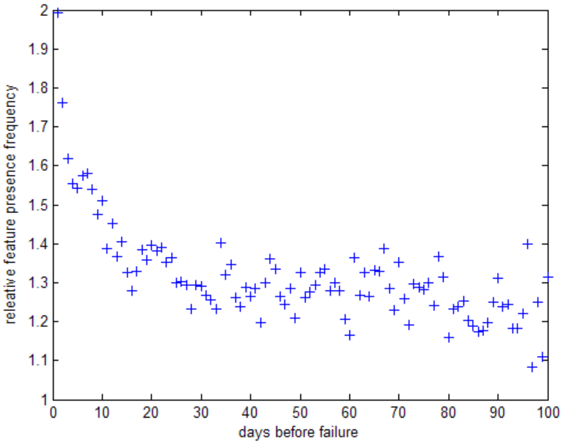


Figure 8: A plot of average (over the fleet) relative frequency of a positive feature vs. time to failure.

On Utility of Single Features An interesting question is whether the effects of an impending failure can be observed over time by looking at only a few features. Our experiments suggest that while the frequencies of some features are in aggregate correlated with approaching failures, they are not strong predictors of failures in individual instances. A plot of the relation between the (relative) frequency of one highly positive feature and the time to a failure is shown in Figure 8. There is a clear upward trend close to the failure date. However, this plot is an aggregate over multiple machines. It turns out that many positive instances do not contain this feature, and the value usually does not change smoothly or in the same direction between consequent days on the same machine. This could be an artifact of how events are logged or it could be related to utilization, which varies from day to day. In either case, we lack additional knowledge to correct for these factors. Thus this feature alone is inadequate for failure prediction. This example illustrated why simple handcrafted rules utilizing only a handful of features do not work well, while machine learning approaches that can use combined even weak signals result in decent performance.

Global vs Local Models Some types of equipment may be configured differently due to specific customer needs. The equipment with the same setup belongs to the same family. Do the equipment logs from different families behave differently? Is the model learned from one family valid for the other ones?

	I	II	III
I	0.43	0.70	0.59
II	0.45	0.87	0.70
III	0.32	0.77	0.71

Figure 9: PM-AUC of cross-testing of family-specific models. E.g. the score on the coordinate (I,II) corresponds to the model that is trained on the data from family I and tested on the data from family II.

To explore similarities between families, we performed hierarchical clustering on the equipment daily bag-of-events features and found that some families behave more similarly to each other than the others. By labeling the same features with their family types and feeding them into a multi-class classifier, we can correctly predict the family of a daily log with more than 80% accuracy on dataset A, which has 28 families. This indicates that there are noticeable differences between families. To explore the predictability of family-specific models on other families, we train a "local" classifier on each family and test on the others. The results (in terms of PM-AUCs) of this experiment on the three largest families in dataset A are summarized in Figure 9. Most of the family-specific models achieve comparable scores in the cross-testing. The fact that the family-specific predictive power is transferable and that known failure cases are very rare suggests that training a global predictive model on data from all the families make sense in our problem domain.

5. INFUSION AND IMPACT

Our approach is currently used to monitor several worldwide fleets of medical equipment. It was implemented and deployed on a SAS platform of a major medical equipment provider. The workflows were built with SAS Enterprise Miner to automate the process of model building and evaluation. Since SAS Enterprise Miner does not include L1-regularized classifiers for feature selection, we ran the proposed algorithm in our development environment to preselect the features which provide the highest PM-AUC score. Only the selected features (usually a few hundred) are used in the SAS Enterprise Miner workflows. Based on the model performance on the held-out historical data, an alert threshold is selected to balance Precision and Recall according to business requirement for a target component. When a problem is identified, an alert is sent to the user. Alerts are reviewed by domain experts to determine the next steps. The system has been deployed in early 2012 and has not been retrained in order to allow a long term real-life performance evaluation. However, it makes sense to retrain the models periodically with the new data, e.g. on a time interval basis or after a number of new failure cases are accumulated.

A real-world evaluation proceeds as follows: no corrective action is taken after receiving an alert during the evaluation period. After that, true positives, false positives and total failures are determined by the component replacement information from the Service Center. Our workflow for predicting a key medical scanner component failure was evaluated over several months on a subset of one fleet (several hundred scanners). The performance was similar to that on the experimental data (Dataset C): it successfully predicted 12 out of 31 failures within 1 week predictive interval and only generated a few (between 0 and 5) false alarms[6]

The business success cannot be measured as easily as the technical one. Evaluation would have to take into account multiple factors, such as cost saving in both spare parts and labor, improved customer satisfaction (e.g. customer return rate) and new business opportunities (e.g. the profit by selling such a service). Evaluating these factors would require intensive involvement of project stakeholders and additional time and resources which is beyond the scope of this data mining project.

6. RELATED WORK

Some recently published research focuses on using log data for anomaly detection. The main advantage is using already existing functionality to improve and better understand systems. Because many systems already include logging functionality exploiting this source of data presents a low cost alternative to augmentation with additional sensors.

Logs can be used to further the understanding of how the system behaves and what are the common patterns. The approach of [15] is aimed at discovering and understanding what the common patterns are, but does not touch upon proactive prevention of errors. In addition to understanding the normal behavior we can also try and detect anomalies. The authors of [26, 25, 24, 27] parse textual log messages and then use PCA to detect anomalies. In our data, however, the changes signaling approaching failure are weaker and noisier (we tried using PCA but did not obtain useful results). Creating event descriptions based on text messages in our logs could be done by clustering [1, 21, 22, 16, 20] but interpreting them requires unavailable domain knowledge. Another option is to use NLP and semantic web techniques to analyze text-based maintenance logs and/or problem ticket data [7, 19] to understand future events or rank events to prioritize repair. Predicting failures on a data stream processing system [13] can be done by observing system's status. This approach uses an ensemble of decision tree classifiers and is applicable in an online setting (i.e. triggering proactive actions when necessary). Taking into account the sequence of logged messages is also possible (we have explored sequence mining, but do not focus on it in this paper). For example, one can use time series of hard drive status messages in Multiple Instance Learning (MIL) setting [17] to predict failures. Finite state automata can be used to model sequential dependencies between messages and detect anomalies [11]. Moreover, temporal information can be incorporated to further improve the detection ability: for example, one could include information about bursts of events [14]. We, on the other hand, use daily aggregates and discard the possibility of fine temporal patterns to simplify the model and reduce the amount of data.

Our data bears some similarities with text represented in bag-of-words format. Machine days can be viewed as documents and events as words describing that day (with their own occurrence frequencies). We can therefore consider some research from the text mining fields as potentially relevant. In many cases, SVMs work well with text derived data which is usually high dimensional and very sparse. Special care has to be taken when dealing with highly imbalanced datasets [5].

Because we can not assign positive label to a specific day before the failure we use multi-instance learning [28, 3, 10] setting to deal with this problem. Training on all individual instances using bag labels works well in many cases [18] but not on our data. Another simple approach is using distance based approaches such as k nearest neighbors (kNN). A more sophisticated approach built on top of kNN is MILES [6] which obtains state-of-the-art performance on many MIL problems by transforming the feature space using distances and then learning a supervised classifier on top of it. Performance of kNN is very poor on our data and MILES suffers from computational complexity while still achieving only mediocre performance. A very successful approach is building ensembles of classifiers [29] which can in many cases improve the results of the base classifier. This idea is partially reflected in our way of doing feature selection, but does not noticeably improve the the performance of the final classifier. MILIS [12] is an approach to MIL that similarly to our non-convex formulation formulation of selecting positive representatives selects prototype instances for positive bags and solves the optimization problem by iterating between instance selection and classifier learning.

7. CONCLUSION AND LESSONS LEARNED

We presented an approach for log-based predictive maintenance. It utilizes state-of-the-art machine learning techniques to build predictive models from log data. Our approach was developed with active involvement from domain experts and was evaluated and shown to be effective by both machine learning and domain standards. It has been deployed by a major medical equipment provider, learning and evaluating predictive models from terabytes of log data, and actively monitoring thousands of medical scanners around the world.

Appropriately formulating the problem by considering the nature of the data and incorporating the domain-based evaluation metrics are the key lessons we learned. Using MIL setting allows us to obtain a good match between data modeling and practical constraints. Data availability and quality determine the quality of the learning process and analytical results. When improving data quality is infeasible, robustness of learning algorithms becomes critical. Additionally, for the domain experts to understand and trust the machine learning algorithms and results, the learned models must be interpretable. We achieve this by building a sparse linear model with L1 regularization. We also notice that using a very small number of features is not sufficient to achieve high performance due to noise and sparsity in data. Thus hand-constructed models do not achieve our level of performance.

Acknowledgments

We thank Patrick Pyka (Siemens AG Healthcare) and Nambi Sridharan (Siemens Medical Solutions USA, Inc.) for insightful domain discussion.

8. REFERENCES

[1] Aharon, M., Barash, G., Cohen, I., Mordechai, E.: One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In: ECML PKDD (2009)

[6] The precise number was not recorded by the customer.

[2] Andrews, S., Tsochantaridis, I., Hofmann, T.: Support vector machines for multiple-instance learning. In: NIPS (2003)

[3] Babenko, B.: Multiple instance learning: Algorithms and applications.

[4] Bach, F.R.: Bolasso: model consistent lasso estimation through the bootstrap. In: ICML (2008)

[5] Brank, J., Grobelnik, M., Milic-Frayling, N., Mladenic, D.: Training text classifiers with svm on very few positive examples. MSR-TR-2003-34 (2003)

[6] Chen, Y., Bi, J., Wang, J.Z.: Miles: Multiple-instance learning via embedded instance selection. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**(12), 1931–1947 (2006)

[7] Devaney, M., Ram, A.: Preventing failures by mining maintenance logs with case-based reasoning. In: Proceedings of the 59th meeting of the society for machinery failure prevention technology (2005)

[8] Dietterich, T.G., Lathrop, R.H., Lozano-Perez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artificial Intelligence **89**(3) (1997)

[9] Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: Liblinear: A library for large linear classification. Journal of Machine Learning Research **9**, 1871–1874 (2008)

[10] Foulds, J., Frank, E.: A review of multi-instance learning assumptions. The Knowledge Engineering Review **25**(1), 1–25 (2010)

[11] Fu, Q., Lou, J.G., Wang, Y., Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis. In: ICDM (2009)

[12] Fu, Z., Robles-Kelly, A., Zhou, J.: Milis: Multiple instance learning with instance selection. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(5) (2011)

[13] Gu, X., Papadimitriou, S., Yu, P.S., Chang, S.P.: Online failure forecast for fault-tolerant data stream processing. In: ICDE (2008)

[14] Lahiri, B., Akrotirianakis, I., Moerchen, F.: Finding critical thresholds for defining bursts. In: DaWaK (2011)

[15] Li, T., Liang, F., Ma, S., Peng, W.: An integrated framework on mining logs files for computing system management. In: KDD (2005)

[16] Makanju, A., Zincir-heywood, A.N., Milios, E.E.: Clustering event logs using iterative partitioning. In: KDD (2009)

[17] Murray, J. F., Hughes, G. F., Kreutz-Delgado, K.: Machine learning methods for predicting failures in hard drives: a multiple-instance application. Journal of Machine Learning Research **6**, 783–816 (2005)

[18] Ray, S., Craven, M.: Supervised versus multiple instance learning: an empirical comparison. In: ICML (2005)

[19] Rudin, C., Passonneau, R.J., Radeva, A., Dutta, H., Ierome, S., Isaac, D.: A process for predicting manhole events in Manhattan. Machine Learning 80, 1–31 (2010)

[20] Taerat, N., Brandt, J., Gentile, A., Wong, M., Leangsuksun, C.: Baler: Deterministic, lossless log message clustering tool. Computer Science - Research and Development **26**, 285–295 (2011)

[21] Vaarandi, R.: A data clustering algorithm for mining patterns from event logs. In: IEEE Workshop on IP Operations and Management (2003)

[22] Vaarandi, R.: Tools and algorithms for mining patterns from event logs. In: NATO ASI Workshop on Mining Massive Data Sets for Security (2007)

[23] Wang, Z., Lan, L., Vucetic, S.: Mixture model for multiple instance regression and applications in remote sensing. IEEE Transactions on Geoscience and Remote Sensing **50**(6) (2012)

[24] Xu, W.: Detecting large scale system problems by mining console logs. Ph.D. thesis, UC Berkeley (2010)

[25] Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Mining console logs for large-scale system problem detection. In: 3rd workshop on Tackling Computer Systems Problems with Machine Learning Techniques, SysML (2008)

[26] Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Online system problem detection by mining patterns of console logs. In: ICDM (2009)

[27] Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.: Using machine learning techniques in console log analysis. In: ICML (2010)

[28] Zhou, Z.H.: Multi-instance learning: A survey. Technical Report (2004)

[29] Zhou, Z.H., Zhang, M.L.: Ensembles of multi-instance learners. In: ECML (2003)