Melody – Expert-Free System Analysis

Sivan Sabato, Elad Yom-Tov and Ohad Rodeh
IBM Haifa Research Labs

1 Introduction

Effective diagnosis and early identification of system problems are crucial to the reliable operation of today's computer systems. In this article we describe the Melody project, where we apply machine learning in several tools that aim to aid in these tasks in IBM System X servers. Several inherent problems present themselves when developing machine learning tools in this domain. These include the difficulty of obtaining labeled examples of system problems, continuous changes in the features describing a system, and the requirement for a low false positive ratio. Our work puts forward several directions for dealing with these problems.

The system's current configuration and status, as well as its event logs, are important resources used by system administrators and support engineers to perform system diagnosis and problem identification. In IBM System X servers, this information is made easily accessible by the IBM Dynamic System Analysis (DSA) application, which collects this information into an XML file[1]. However, a typical DSA XML file includes thousands of data items and tens of thousands of event log messages. Thus, it is impossible to manually read through all this information.

The Melody project aims to assist system administrators and support engineers by providing tools to analyze the data collected by DSA. A common approach to system analysis is to use manually encoded information, such as system specifications and alert rules, to provide predefined indications about the inspected system. However, this approach severely limits the usability of the tools over time; it is usually not feasible to keep the specifications and rules up-to-date as systems change, versions are upgraded, and system use-models evolve. For these reasons, Melody is based on a purely data-driven approach. Information manually encoded by domain experts is precluded. Only information that can be learned from available data may be used. Furthermore, tools must be designed to adapt automatically to changes in system populations over time. This is a challenge to the underlying algorithms, as the tools must re-train periodically as the system population changes, and they must do so without human intervention. This implies the following restrictions on any algorithm employed by Melody:

- Only labels that can be reliably retrieved for new data may be used. This precludes the use of targeted manual labeling for a pre-determined dataset.
- The features describing the system cannot be assumed to be known in advance; the set of possible system components, messages, and configuration attributes evolve constantly.
- The tools must provide a good estimation of the quality of the results for each analyzed system. To maintain the trust of the users, it is better to provide no results on some occasions than to provide low-quality results.

Melody uses DSA XML files that have been recorded in the support center as a sample of the population of operational systems. Thousands of such files, representing a similar number of systems, are collected by DSA each month. However, as is often the case with system related data, reliable information describing the problem in the system is usually not available.

Two Melody tools that we have developed, the Configuration Analyzer and the Event Log Analyzer, are already in use in IBM System X support centers around the world. We present these tools and preliminary results for an additional tool, the Event Rules Detector.

2 Configuration Analyzer

The input to the Configuration Analyzer is a description of the system configuration, which is provided as a set of component descriptions. Each component description corresponds to a hardware or software component in the system and is described by a set of attributes and values. For instance, a component may be one of the system's hard drives and its description may include attributes such as the hard drive model, the controller type, and the current size of the free space. The set of attributes included in a component description depends on the type of component. A typical system has hundreds of components with a total of thousands of attribute-value pairs. Many of the attributes have categorical values, while some attributes are numerical or free strings. The set of possible values for each attribute is not known in advance. Likewise, the number of possible values for an attribute is unknown and ranges from two to a practically unbounded number – for example, in attributes that can include user-typed strings.

When applying machine learning algorithms to system configuration descriptions, one of the first challenges we face is how to model the descriptions for use with standard algorithms. Many machine learning algorithms assume that each data point is represented by a feature vector. This becomes more complex for system configurations since different systems may have different types and numbers of components, and there is no trivial mapping between components of one system and components of another. For example, some systems have four processors while other systems have two. Different drivers may exist in different systems, with each driver described by a different set of attributes.

A representation similar to the text analysis bag-of-words approach may be considered. In this representation, the features indicate the existence of a specific value for a specific attribute. This approach has the disadvantage of creating an abundance of features that may cause overfitting and high computational overhead. In the Melody Configuration Analyzer, we take another approach, where each type of component defines a separate dataset and each component of this type represents a single data point. A system with four processors thus contributes four data points to the processor dataset.

[1] The format of the file conforms to the Common Information Model, see http://www.dmtf.org/standards/cim/

This approach renders the different datasets manageable, although it should be taken into account that the examples in each dataset are not statistically independent. In addition, it is not possible to identify relationships between component types within this approach.

The tool identifies attribute values that are anomalous in the analyzed system as compared to the training set systems, and displays only those attributes to the user. This summarization technique is based on the premise that even though most of the systems in the training set have some faults, the faults and their causes are diverse; each specific problem is present only in a small fraction of the systems. Therefore, it can be assumed that sub-configurations that are relevant for a system problem are ones that are not common among the systems in the training set. For each attribute, the tool generates a classifier that identifies anomalous values based on the training set. The classifiers are built so that the marginal probability of each classifier detecting an anomaly is no larger than a predetermined threshold. This guarantees that the expected number of items displayed in a system's configuration summary is limited, while still allowing variance in the number of anomalies found in each system.

The presence of many categorical features with an unknown number of values in a system's configuration requires special care. In the anomaly detection setting, we must estimate the missing probability mass [1] to ensure that the anomaly threshold is correct. In a supervised setting, feature selection and classification algorithms that support these types of features are needed (see, for instance, [3]). Automatic recognition of feature types is also required to ensure that the correct technique is applied to each feature.

The Configuration Analyzer produces useful results, as the example in Fig. 1 demonstrates. Adding supervision to this process may further improve the results. While it is not feasible to obtain labels for all the systems in a dataset, it may be possible to use a hybrid approach, where the parameters of the anomaly detection mechanism are learned as 'meta-features'. Interactive responses from users of the tool indicating the helpfulness of the anomaly results may allow us to tune the algorithm parameters. It might be possible to model this setting as a reinforcement learning problem [5].

3 Event Log Analyzer

The contents of the systems event logs are an important part of the system's description collected by DSA. Some examples include the Windows event logs in Windows and the syslog in Linux. Event logs record the messages emitted by various components of the system during its day-to-day operation. Emitted messages may be informational or they can indicate a problem in the system, whether trivial or more serious. System logs usually record a large number of messages, most of which are not relevant for problem identification or diagnosis. It is usually impossible to manually read through all the messages to identify the relevant ones.

Given the event logs from a specific system, the Event Log Analyzer generates a summarized ranked view of this log. The messages are clustered into mutually exclusive sets that correspond to message type profiles. Message groups are then ranked by how unexpected it is for a system of this type to have this number of messages in this message group. Finally, messages are displayed in order of rank (see Fig. 2). The probability of having this number of messages in the message group is estimated using the DSA dataset. A comprehensive presentation of the Event Log Analyzer can be found in [4].

4 Event Rules Detector

The Event Rules Detector automatically identifies decision rules that typify system problems. Although labels that indicate the presence of a problem in a system are not readily available, we obtain labels by assuming that user behavior implies a labeling. In the context of the IBM System X support centers, if a DSA XML file was collected from a system, it can be assumed that the system was faulty at the time of collection. This is a very noisy label: users sometimes mistakenly identify a system as faulty, DSA may be activated only some time after a fault has developed, and it may be activated when no fault has been identified. However, the experiment described below shows that this assumption can still be used to create valuable decision rules.

In the experiment a dataset of event logs from hundreds of systems was used. We built a feature vector for each calendar day in which the system emitted messages. The features are counts of the number of messages of each type found in the log in each of several time windows. Thus, one feature may be the number of system restarts during the current day, another feature is the number of restarts during the last two days, etc.

Each feature vector was labeled by assuming that the most recent day recorded in the event log, i.e., the day the DSA application was activated, indicated a faulty system, and that all the days that are more than two weeks earlier than the most recent day indicated non-faulty systems. We built a decision tree for separating these feature vectors using the C4.5 algorithm [2], and rules that identify faulty systems with high probability (in our case, more than 99% correct) were extracted from the tree. Preliminary indicators show that this method finds meaningful rules from the data. An example is shown in Figure 3. These rules could be applied for diagnosing a problem after it has occurred, or for early identification of imminent system problems.

References

[1] D.A. McAllester and R.E. Schapire. On the convergence rate of good-turing estimators. In *COLT*, 2000.

[2] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[3] S. Sabato and S. Shalev-Shwartz. Prediction by categorical features: Generalization properties and application to feature ranking. In *COLT*, 2007.

[4] S. Sabato, E. Yom-Tov, A. Tsherniak, and S. Rosset. Analyzing system logs: A new view of what's important. In *SysML*, 2007.

[5] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

A Appendix: Figures

Score	Type	CIM Type	Property Name	Value	Frequency	Most Common Values	Percentage
83	Uncommon value	Logical Disk	DiskQuotaStatus	1	0.84%	0	96%
75	Uncommon value	Logical Device [DeviceClass: scsiadapter]	OtherIdentifyingInfo	{ emulex lightpulse hba - storport miniport driver }	1.27%	{ }	98%
51	Uncommon value	System Service Information [Name: snmptrap]	Version	5.2.3790.1830 (srv03_sp1_rtm.050324-1447)	2.46%	5.2.3790.0 (srv03_rtm.030324-2048)	66%
						5.00.2195.6601	29%
46	Uncommon value	PCI Device	ProgrammingInterface	128	2.71%	0	32%
						32	23%
						138	21%
						16	13%
						130	7%
37	Uncommon value	USB Device	USBVersion	256	3.16%	272	48%
						512	22%
						110	15%
						200	11%

Figure 1: An example output of the Configuration Analyzer

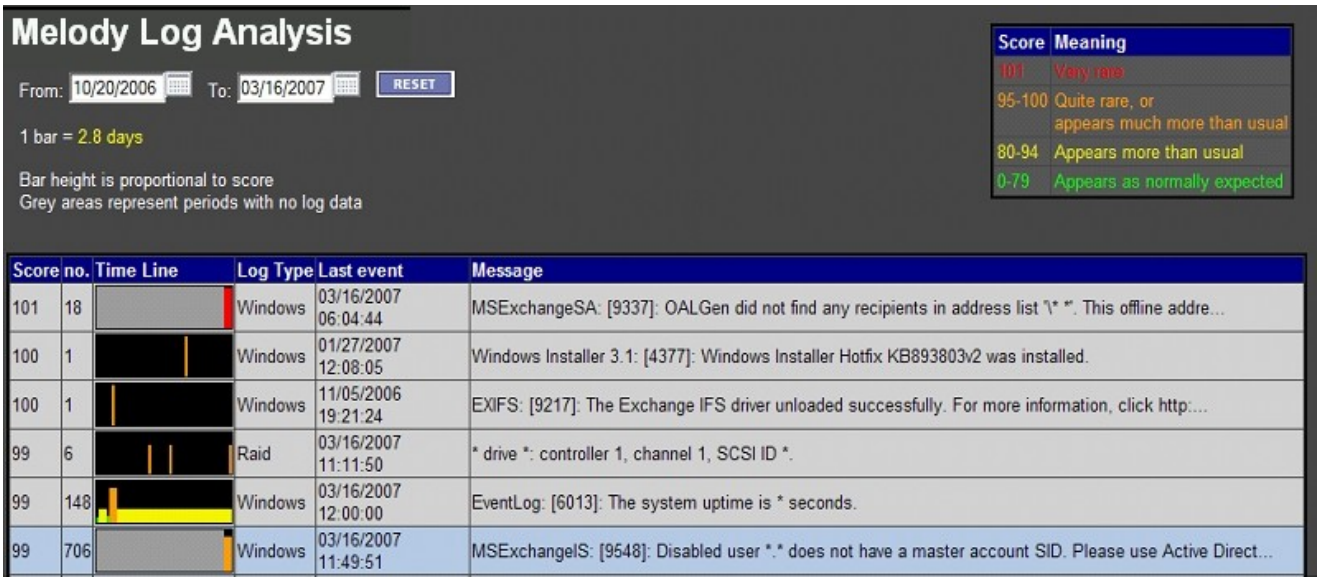

Figure 2: An example output of the Event Log Analyzer

Rule 1 (System log):
$(eventID = 26(t-26) > 58)$ & $(Source = Print(t-21) \leq 620)$
Explanation: Too many failed writes into a system which is not a print server.

Rule 2 (Application):
$(eventID = 17055(t-22) \geq 873)$ & $(eventCount(t-22) \leq 4854)$
Explanation: SQL server keeps failing

Figure 3: Examples of rules generated by the Event Detection Rules module.