Abstract

Logs which carry valuable runtime information of modern systems, are widely utilized by developers (and operators) in system development and maintenance. Due to the ever-increasing size of logs, data mining models are adopted to help developers extract system information. To facilitate engineers perform better automated log-based problem identification, in this project, we propose a log classification based anomalous execution sequence detection framework, in which we first employ log parser to abstract the log messages, and recover log execution sequences. Then we train classification model using log sequences which contain labels indicating normalizes or anomalies. When we get a new log sequence without a label, we can use our well-trained classifier to determine its label. To validate the effectiveness of our framework, we will carry experiments on real Hadoop logs with well-established anomaly labels and adopt 10-fold cross validation for evaluation.

1 Introduction

With advancement in science and technology, computing systems are becoming increasingly more complex and scalable. Large-scale computing systems such as Hadoop are commonly used in today’s big data era. These systems often generate huge amounts of system logs for troubleshooting. The log messages are usually semi-structured text strings, which are used to record events or states of interest. In general, when a job fails, engineers can examine recorded log files to gain insight about the failure through their domain knowledge. Logging is particularly important for large-scale online services running in a Big Data environment with multiple clusters of servers and data centers, where other software debugging techniques are difficult to be applied. So log-based problem identification has been widely used in practice.

Although important, log-based problem identification is not easy. Traditionally, when a service failure occurs, engineers identify problems by searching for “erroneous” jobs in the generated logs. They perform simple keyword search (such as “kill”, “fail”, “error”, and “exception”) of logs that may be associated with the failure. Figure[1] shows an example with a normal log message and an abnormal log message which may be used for problem identification. Due to the increasing scale and complexity of computing systems, the number of generated logs could be quickly overwhelming. Clearly, it can be very time consuming and error prone for a human operator to diagnose system problems by manually examining a huge amount of log messages.

*Data mining project report, department of computer science and engineering, CUHK, Spring 2015.
To facilitate engineers perform better automated log-based problem identification, in this project, we propose a log classification based anomalous execution sequence detection framework, which consists of a training phase and a testing phase. Before training and testing, we first employ log parser to abstract the log messages, and recover log execution sequences. Then we train classification model using log sequences which contain labels indicating normalizes or anomalies. In testing phase, when we get a new log sequence without a label, we can use our well-trained classifier to decide whether or not its an anomaly. To validate the effectiveness of our framework, we will carry experiments on real Hadoop logs with well-established anomaly labels and adopt 10-fold cross validation. In this project, we employ four different kinds of classifiers and will compare their performances.

2 Methods

In this section, we will briefly introduce the framework of our project and the background about Log Parsing. Then we will formulate the log classification problem and describe four different classification methods in details, i.e. Naive Bayes, Single Layer Perceptron, SVM and Decision Tree.

2.1 Framework

The overall structure of our proposed method is shown in Figure 2. We can see that there are mainly two phases: training phase and testing phase. Training phase uses labeled data to train classifier’s parameters. Testing phase predicts the label of unlabeled data. And in every phase, there are Log Parsing and Log Vectorization procedures to generate system execution sequences. Considering that a single type of log messages is often not sufficient to pinpoint the problem, but the relationship among a group of log messages such as correlation and relative frequency, can be used for problem identification. So in our problem, the input to classifiers is log sequence rather than log message, in whose context it is much easier to identify a problem.

The major steps shown in Figure 2 are as follows. We describe them in detail in this section:

Log Parsing: A log message usually contains two types of information: constant strings and parameters. Parsing is to remove those specific parameters and form abstracted and generic log events. The details about Log Parsing are given in Section 2.2.
Log Vectorization: With the same task ID, a group of log events can be linked into a log sequence, representing a system task. Further, we can turn each log sequence into a vector, in which each dimension represents the frequency of a specific type of log event. In this way, our input for classifier is an \( n \times m \) matrix with \( n \) representing the number of log sequences and \( m \) representing the number of log events.

Classification: We adopt four different kinds of frequently-used Classification methods by using a standard python package. And 10-fold cross validation is employed in our project since all our log sequences have labels. And there will be a performance comparison. More details about the classifiers are described in Section 2.3.

2.2 Log Parsing

Figure 3 illustrates an overview of log parsing. The raw logs, as shown in the figure, contain ten lines of log messages extracted from HDFS log data on Amazon EC2 platform [15]. The logs are unstructured data, with timestamps and raw message contents (some fields are omitted for simplicity of presentation). In real-world cases, a log file may contain millions of such log messages. The goal of log parsing is to distinguish between constant part (fixed plain text) and variable part (e.g., blk ID in the figure) from the log message contents. Then, all the constant message templates can be clustered into a list of log events, and structured logs can be generated with each log message corresponding to a specific event. For instance, the log line 2 is transformed to “Event2” with a log template “Receiving block * src: * dest: *”. The output of a log parser involves two files with log events and structured logs. Log events record the extracted templates of log messages, while structured logs contain a sequence of events with their occurring times. Finally, the structured logs after parsing can be easily processed in log mining methods, such as anomaly detection and deployment verification [13].

Tool: SLCT (Simple Logfile Clustering Tool) [14] is, to the best of our knowledge, the first work on automated log parsing. The work also released an open-source log parsing tool, which has been widely employed in log mining tasks. So in our project, we adopt the released SLCT package for Log Parsing.

---

**Figure 3: Overview of Log Parsing**
2.3 Classification Model

In this Section, we will introduce the details about our classification models, including Naive Bayes, Single Layer Perceptron, SVM and Decision Tree.

2.3.1 Single Layer Perceptron

Single Layer Perceptron is a Neural Networks model with one input layer and one output layer \[8\]. The algorithm is simple and suitable for large scale learning, which is not regularized (penalized) and updates its model only on mistakes. Figure 4 shows the model of Single Layer Perceptron.

\[
\text{net} = x_1 w_1 + x_2 w_2 + ... + x_n w_n + b, \quad y = f(\text{net})
\]

In SLP, we have \( \text{net} = x_1 w_1 + x_2 w_2 + ... + x_n w_n + b \), and \( y = f(\text{net}) \). Usually the \( f(\text{net}) \) is a Step or Sign Function. Learning of Perceptron is done by \( w_i \leftarrow w_i + \alpha(t - y)x_i \).

2.3.2 Gaussian Naïve Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naive” assumption of independence between every pair of features. Given a class variable \( y \) and a dependent feature vector \( x_1 \) through \( x_n \), Bayes’ theorem states the following relationship:

\[
P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)}
\]

Using the naive independence assumption that

\[
P(x_1, ..., x_n|y) = \prod_{i=1}^{n} P(x_i|y)
\]

Since \( P(x_1,...,x_n) \) is constant given the input, we can use the following classification rule:

\[
P(y)P(x_1, ..., x_n|y) \propto P(y) \prod_{i=1}^{n} P(x_i|y)
\]

\[
\Rightarrow \hat{y} = \arg \max_y P(y) \prod_{i=1}^{n} P(x_i|y)
\]

and we can use Maximum A Posteriori (MAP) estimation to estimate \( P(y) \) and \( P(x_i|y) \); the former is then the relative frequency of class \( y \) in the training set. Here, the likelihood of the features is assumed to be Gaussian:

\[
P(x_i|y) = \frac{1}{\sqrt{2\pi \sigma_y^2}} \exp(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2})
\]
The parameters $\sigma_y$ and $\mu_y$ are estimated using maximum likelihood.

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

![Figure 5: Support Vector Classification](image)

### 2.3.3 Linear SVC

A linear Support Vector Classification (SVC) \cite{2} constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks, as Figure 5 shows. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. As the training data are linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called "the margin" and the maximum-margin hyperplane is the hyperplane that lies halfway between them. So the problem of linear SVC can be formulated as follows:

$$
\min_w \frac{1}{2} \| w \|^2 + C \sum_i \xi_i \\
\text{s.t. } y_i (w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0
$$

(6)

To solve the problem, we often adopt the Stochastic Gradient Descent technique, and parameter $C$ can be set using cross validation. Kernel Trick is often used for non-linear SVM.

### 2.3.4 Decision Tree

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item’s target value. In this project, we use CART (Classification and Regression Trees) algorithm to implement decision tree classification. Classification and regression trees (CART):

- Impurity metric:
  $$
  Gini(n) = 1 - \sum_i p(c_i | n)^2
  $$
  (7)

- Construct binary tree using the feature and threshold that yield the largest information gain at each node
  $$
  Gain(X, T) = Info(T) - Info(X, T)
  $$
  (8)

- Nodes are expanded until all leaves are pure
  $$
  Gini(i) = 0 \quad \forall i \in LeafNodes
  $$
  (9)

- Pruning to avoid over fitting: setting the minimum number of samples required at a leaf node or/and setting the maximum depth of the tree
3 Experiments

In this section, we will describe our experiments for evaluating four different Log Classification methods. More specifically, we will give the detail setting of model parameters and comparison of classification results.

3.1 Experiments Design

Data: Our log data sets are HDFS logs collected in paper [15] from a 203-node cluster on Amazon EC2 platform, with well-established anomaly labels through domain knowledge. The data set contains totally 11,175,629 (11 million) raw log messages. After Log Parsing, we can get 575,139 log sequences with 29 kinds of log events, as table 1 shows.

| Events | 29 |
| Log sequences | 575,139 |
| Anomalies | 16,838 |

Evaluation: As all of our log sequences have labels to indicate whether or not its an anomaly, so we do not have testing data. In this way, we employ 10-fold cross validation to evaluate our classification models. For each model, we adopt Precision, Recall, Fmeasure and Accuracy [11] for evaluation, which are all common-used metrics.

Machine: All experiments were performed on a single PC equipped with Windows7 Ultimate 64-bit, which has Intel(R) Core(TM)i5-3210M CPU and 6 GB DDR3 1600 RAM.

Tool: Scikit-learn This is a simple and efficient Python package for data mining and data analysis, which is built on NumPy, SciPy, and matplotlib. It provides us with all kinds of machine learning algorithms in packages including classification, with user-friendly interfaces. So in our project, except for SLP, we all use Scikit-learn to perform classification. All we need to do is to set the parameters.

3.2 Experiments Results

To begin with, figure 6 directly gives us an overview of results on four different classification models, and we will describe them one by one in the following part.

3.2.1 Single Layer Perceptron

As Section 2.3.1 indicates, the updating rule for Single Layer Perceptron is:

\[ w = w + \alpha \ast (t - y) \ast x \]

So, in our experiment, we simply set the learning step size \( \alpha = 0.8 \).

3.2.2 Gaussian Naïve Bayes

There is “Naïve” assumption of independence between every pair of features. The likelihood of the features is assumed to be Gaussian:

\[ P(x_i|y) = \frac{1}{\sqrt{2\pi}\sigma_y^2} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \]  

Using maximum likelihood to estimate \( \sigma_y \) and \( \mu_y \). The results indicate that Gaussian Naïve Bayes can not beat over other three. However, comparing with other three models, Gaussian Naïve Bayes needs the shortest time to train model and has the best performance on recall.

3.2.3 Linear SVC

For Linear Support Vector Classification, the parameter setting is:

- loss functions : hinge
• Penalties : L2 norm ; C=1.0
• Stopping criteria : tolerance = 1e-4

Linear SVC has quiet a good performance in classification job 9(a). However, its model training process is time-consuming 9(b) that exhaust 18 times more than the average of other three.

3.2.4 Decision Tree

Experiments show that CART owns the optimal performance on our classification job as 9(a) indicates, with not significantly too much train time in 9(b).

At the same time, it generate an understandable decision structure which can give us more information about classification. Like figure 7 shows that, $y_0$ comes from normal log and $y_1$ come from anomaly log. And we also can get more information about different feature(splitting node), for example, different color represents different impurity.

3.3 Classification Results Compare

Here we present the overall comparison of the four models in Table 2 and Figure 8.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.8907</td>
<td>0.9995</td>
<td>0.9420</td>
<td>0.9964</td>
<td>26.2431</td>
</tr>
<tr>
<td>SLP</td>
<td>0.9798</td>
<td>0.9875</td>
<td>0.9837</td>
<td>0.9990</td>
<td>59.1227</td>
</tr>
<tr>
<td>SVM</td>
<td>0.9849</td>
<td>0.9960</td>
<td>0.9904</td>
<td>0.9994</td>
<td>966.4003</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.9995</td>
<td>0.9982</td>
<td>0.9988</td>
<td>0.9999</td>
<td>71.8275</td>
</tr>
</tbody>
</table>

4 Related Works

There already exists many works on log-based problem identification for system diagnosis. These works retrieve useful information from logs and adopt data mining and machine learning techniques to analyze the logs for problem detection and diagnosis. For example, Lou et al. [9] mine invariants
(constant linear relationships) from console logs. A service anomaly is detected if a new log message breaks certain invariants during the system execution. Xu et al.\cite{Xu2015} preprocess the logs and detect anomalies using principal component analysis (PCA). LogEnhancer\cite{Yi2016} aims to enhance the recorded contents in existing logging statements.

5 Conclusion

From figure\ref{fig:performance}(a) we can get the log classifying performance order: Naive Bayes < Single Layer Perceptron < SVM < Decision Tree. On the other hand, the figure\ref{fig:performance}(b) indicate the efficiency order: Naive Bayes < Single Layer Perceptron < Decision Tree < SVM.

In details, each classifier has its own feature. Gaussian Naive Bayes learns fastest with relative bad performance. Oppositely, Linear SVC learns extremely slow with good performance. And Single
Layer Perceptron is in the middle between them. Specifically, Decision Tree does much well on both classification performance and efficiency.

6 Future Works

In practice, we may face much larger log dataset which would be much difficult to extract critical patterns inside the logs, for instance, the logs of Google work station center. Considering this case, we definitely need more powerful discriminative model that owns large capacity to handle it. Recently deep neural networks (DNN) [4, 5, 7] have achieved significant success in variety of machine learning tasks spanning from computer vision, natural language processing to robots control [6, 3, 12, 10] which demonstrate its powerful capability on supervised learning tasks. In the future, we will deploy DNN to learn to capture unique pattern of different event logs and then utilize these features to perform the classification task.

References


