Structural and Temporal Patterns-Based Features

Venkatesh-Prasad Ranganath
Microsoft Research, India
rvprasad@microsoft.com

Jithin Thomas
Microsoft Research, India
t-jithit@microsoft.com

***Abstract*—In this paper, we propose a data transformation pattern to transform sequential data into a set of binary/categorical features and numerical features to enable data analysis. These features capture both structural and temporal information inherent in sequential data.**

I. Categorization

Assuming data analysis consists of four phases: *data cleansing, data transformation, data processing,* and *result validation*, the proposed pattern is a data transformation pattern.

II. Intent

Extract features from sequential data composed of events for the purpose of data analysis. The features should *capture event ordering information* inherent in the data. If the events are structured, the features should *capture structural information* inherent in events.

III. Motivation

Consider a scenario where we have a set of logs from a web server under various loads and we want to identify similar loads on the web server. If similar loads result in similar logs, then we can compare the logs for similarity, e.g.compare the frequency distribution of events in logs. An immediate extension of this scenario is to detect if a new load (based on the corresponding log) is similar to any of the previously observed loads.

Another scenario is when we have (failing) logs from a web server that is functioning with errors and we would like to identify the possible faults/causes. If the logs capture information about faults (or failures or errors) and we have (non-failing) logs from the same web server while functioning without errors, then we can compare failing logs and non-failing logs to identify "features" that are unique to failing logs. The comparison could be based on simply on the presence/absence of events and event orderings.

Above scenarios are specific instances of *classification, clustering,* and *trace comparison* problems that often arise during data analysis. While classic off-the-shelf clustering and classification algorithms can be used to address these problems [2], their success would depend on the richness of the features extracted from the data. Hence, feature extraction is an important task during data analysis, and patterns-based feature extraction is a way to accomplish this task.

IV. Description

Most often, data is a composition of instances of various patterns. Hence, given a set of patterns exhibited by a data set, *binary/categorical features* can be constructed to indicate if a pattern is exhibited by a datum.

A. Structural Patterns-Based Features

Given the structured data {*x=3, y='foo', z=3*} where *x, y,* and *z* are attributes of the data, all non-empty subsets of attributes along with their values {*x=3, y='foo', z=3*}, {*x=3, y='foo'*}, {*y='foo',z=3*}, {*x=3, z=3*}, {*x=3*}, {*y='foo'*}, and {*z=3*} are valid abstractions of the data and they capture structural information about the datum. Hence, such abstractions can be viewed as *structural patterns* exhibited by the data.

With the set of structural patterns observed in a data set, *binary/categorical features* can be trivially constructed. Such features can also be constructed for sequential data by considering the structural patterns exhibited by events in sequential data.

B. Temporal Patterns-Based Features

Consider the following trace of events A, B, C, and D.

```
A A C A B D B C B D
```

This trace can be viewed as a composition of temporal patterns that capture event ordering information.

```
A A - A B - B - B -
- - C - - D - C - D
```

Specifically, based on the above projections of the trace, the trace can be viewed as a composition of temporal patterns: $A \overset{*}{\rightarrowtail} B$, $A \overset{*}{\leftarrowtail} B$, $C \overset{a}{\rightarrowtail} D$, and $C \overset{a}{\leftarrowtail} D$ where the patterns are defined as follows.

- $X \overset{*}{\rightarrowtail} Y$ ($Y \overset{*}{\leftarrowtail} X$) patterns denote every occurrence of event X will be followed (must be preceded) by an occurrence of event Y. These temporal patterns are referred to as *eventually patterns*, and *fopen* $\overset{*}{\rightarrowtail}$ *fclose* and *fopen* $\overset{*}{\leftarrowtail}$ *fclose* are examples of such patterns.
- $X \overset{a}{\rightarrowtail} Y$ ($Y \overset{a}{\leftarrowtail} X$) patterns denote every occurrence of event X will be followed (must be preceded) by an occurrence of event Y without any intervening occurrence of event X. These temporal patterns are referred to as *alternation patterns*, and *lock* $\overset{*}{\rightarrowtail}$ *unlock* and *lock* $\overset{*}{\leftarrowtail}$ *unlock* are examples of such patterns.

Instead of considering entire events, abstractions of events can be considered to construct temporal patterns. For more

details about these sorts of structural and temporal patterns, please refer to [3].

V. Considerations

A. Applicability

Structural patterns-based feature extraction works well when

- events are structured,
- not all parts of events are equally relevant, and
- correlations between attributes and their valuations in events are relevant.

Similarly, temporal patterns-based feature extraction works well with sequential data and the ordering between events is relevant.

B. Cost (Complexity)

Observe that, an event with 10 attributes can yield $2^{10}-1 = 1023$ structural patterns and $2^{20} \approx 1046529$ linear binary temporal patterns of a kind (e.g.binary eventually follows patterns). With this sort of explosion in patterns, the use of this pattern and the subsequent data analysis can be expensive. An effective way to curb this explosion (and associated cost) is to use domain knowledge to limit the number of attributes considered to generate the abstractions of events. This results in fewer number of abstractions per event. For example, if only 5 out of 10 attributes are relevant, then the number of structural and linear binary temporal patterns comes down to $2^5 - 1 = 31$ and $2^{10} \approx 961$, respectively.

Another technique to curb this explosion is to use domain knowledge to abstract values of attributes and limit the number of unique abstractions considered during analysis. For example, while dealing with software trace data, it might suffice to abstract 32-bit pointer values as `NULL` and `non-NULL` values, i.e. $2^{32} - 1$ unique values are reduced to 2 unique values!

C. Choice of Patterns

While most sequential data sets exhibit both structural and temporal patterns, there can be scenarios in which only structural patterns-based features might suffice. Similarly, there can be scenarios that demand temporal patterns-based features. However, in many scenarios, the best feature vector for a data set will be a combination of features based on both sorts of patterns. So, depending on the application scenario (and the domain), an appropriate combination of features should be identified. For example, when comparing two data sets in terms of unique features, a combination of features based on all structural patterns along with features based only on temporal patterns that involve structural patterns that are common to both data sets (ignoring unique temporal patterns that involve unique structural patterns) would form a good feature set.

In a similar vein, the choice of using patterns involving abstractions will yield different sets of features for a data set and this choice should be driven by the domain and the application scenario.[1]

[1]We explored both these choices when we used patterns-based features to perform compatibility testing as described in [5].

D. Alternative Patterns

In this exposition, we have considered the simple linear binary temporal patterns defined in [3]. As alternatives, n-grams [4] and finite state machines can be considered as well.

E. Quantitative Features

Most often, observed patterns are associated with a numerical/statistical property local to data (e.g. frequency, minimum/maximum distance between events participating in temporal patterns). So, we can construct *quantitative features* that represent statistical properties of patterns observed in the data.

F. Presentation

A set of structural patterns along with the $\subseteq$ relation form a partially ordered set. Hence, while presenting structural patterns-based features to humans, only features based on structural patterns that are maximal/minimal elements of the partially ordered set can be presented to reduce information overload. For example, given a set of unique structural patterns-based binary features $\{f_1, f_2, f_3, f_4\}$ such that only $f_1 \subseteq f_4, f_2 \subseteq f_4$, and $f_3 \subseteq f_4$ (where $f_x \subseteq f_y$ denotes the set of structural patterns corresponding to f_x is a subset of the set of structural patterns corresponding to f_y), either $\{f_4\}$ or $\{f_1, f_2, f_3\}$ can be presented without loss of information.

A similar technique can be adopted to trim a set of temporal patterns-based features. Also, these techniques can be used as feature filters while processing patterns-based features.

VI. Example

The USB 3.0 protocol driver stack in Windows 8 was a clean room implementation. To ensure the USB 3.0 protocol driver stack supported existing USB 2.0 devices, the team had to test if the behavior of the USB 3.0 protocol driver stack was similar to the behavior of the USB 2.0 protocol driver stack in Windows 7 when both stacks were servicing the same USB 2.0 device. Besides testing with various devices to uncover incompatibility issues observable via device failures, they also employed patterns-based trace comparison—logs of traffic between USB 2.0 client device drivers and USB 3.0 and 2.0 protocol driver stacks were compared in terms of their structural and temporal patterns-based features (akin the use case scenario mentioned in Section III). This comparison uncovered compatibility issues even when devices under test were functioning without errors.

An example of compatibility issues that can be captured as patterns is the USB 3.0 protocol driver stack completed isochronous transfer requests at `PASSIVE_LEVEL` interrupt request level while USB 2.0 driver stack completed such requests at `DISPATCH_LEVEL` interrupt request level. This issue was captured as a structural pattern.

The details of this effort can be found in [5]. The tool used to mine structural and temporal patterns in this effort can be found at [1].

REFERENCES

[1] Tark: Mining linear temporal rules. http://research.microsoft.com/en-us/projects/tark/, 2011.

[2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[3] David Lo, Ganesan Ramalingam, Venkatesh Prasad Ranganath, and Kapil Vaswani. Mining quantified temporal rules: Formalism, algorithms, and evaluation. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, 2009.

[4] Christopher D. Manning and Hinrich Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[5] Venkatesh Prasad Ranganath, Pradip Vallathol, and Pankaj Gupta. Compatibility testing via patterns based trace comparison. In *Microsoft Technical Report*, 2012.