Finding Structure in Streams

Paul R. Cohen and Tim Oates
Experimental Knowledge Systems Laboratory
Department of Computer Science, LGRC
University of Massachusetts
Box 34610
Amherst MA 01003-4610
cohen@cs.umass.edu, oates@cs.umass.edu

Abstract

Finding structure in streams (series of categorical data) is an important task. Consider a patient in an intensive care unit, where monitors record different aspects of the patient's condition. There is clearly utility in determining how current values of those monitors are indicative and predictive of various features of the patient's health. We present four algorithms for finding different types of structure in one or more streams. The first algorithm finds predictive relationships among the tokens in a single stream. The next two algorithms find predictive relationships between the values in multiple streams over a fixed time interval; one is a batch algorithm and the other is incremental. Finally, we present a representation for ongoing processes in streams, called fluents, and an algorithm for finding fluents and associations among them in multiple streams.

Keywords

dependency detection, time series, fluents

Introduction

Streams are series of categorical data; for example, the letters in this sentence constitute a stream, beginning "`S t r e a m s   a r e . . .`". Elements in streams, called tokens, can be complex objects provided they are treated as atomic, that is, they are not decomposed when streams are analyzed. Thus, letters, morphemes and words are three kinds of tokens in a paragraph stream. Some data, such as text, is naturally represented as a single stream, but we often have several synchronized streams. Suppose a baby is lying in its crib, occasionally waving a rattle, which produces an alternating high-low tone:

```
Wave  W W W W W O O O O O O O W W W W W W W W O O ...
Tone  H H L L H H X X X X X X X H L H H H L H H L ...
Gaze  R R R C C C C C C R R R R R R R R R C C C C ...
```

Each column or multitoken represents the state of three streams at a particular time; for example, $< WHR >$ is the first multitoken. These streams contain a lot of structure. First, the rattle makes a noise (High or Low tone) when it is waved (W), and the sound stops (X) almost simultaneously with the cessation of waving (O). Second, Baby's gaze is directed to the rattle (R) when it starts making a noise, but is diverted to the crib (C) after a while. Third, orienting the gaze to the rattle precedes waving it. Fourth, the sound of a rattle is alternating H and L tones, not of identical duration, but not longer than three time steps or shorter than two. Finally, once the baby starts waving, it will wave for several time steps.

In the remainder of this paper we describe four algorithms for finding structure in streams of data. The first finds structure in a single stream of data, and was successfully applied to facilitating debugging of a planning system by analyzing program execution traces. The second, called MSDD, attacks a more general problem than the first algorithm by finding structure in batches of data composed of multiple streams. The third algorithm, called IMSDD, was designed to perform the same task as MSDD but in an incremental manner, without requiring access to a complete data set prior to beginning the search for structure. Finally, we discuss a representation for ongoing processes, called fluents, and describe an algorithm for finding fluents and associations among them in multiple streams of data.

Finding Structure in a Single Stream

The first algorithm was developed by Adele Howe and Paul Cohen for finding dependencies in a single stream (Howe and Cohen, 1995). Consider the Wave stream, above. It contains two tokens, W and O, which apparently are not distributed uniformly. A simple G test on a contingency table tells us that W follows itself more often than we'd expect by chance under the uniform distribution hypothesis. The contingency table is shown in Figure 1.

	W	$\overline{W}$	Totals
W	11	2	13
$\overline{W}$	1	7	8
Totals	12	9	21

Figure 1: Example contingency table from the Wave stream shown above.

The Wave stream contains 11 cases of W following itself (i.e, W is both predecessor and successor), 2 cases of W followed by something other than W (denoted $\overline{W}$), one case of $\overline{W}$ followed by W and seven cases of $\overline{W}$ followed by $\overline{W}$. The G statistic for this table is 11.49, p = .0012. In short, occurrences of W are not independent of immediately prior occurrences. It would be easy to introduce a lag into the analysis to find dependencies between one token and another after, say, five time steps, and Howe has designed an adaptive algorithm to find the most predictive lag for dependencies (Howe, 1995). Note also that the average length of a "run" of W's is just the first row margin (13) divided by the cell 2 count, that is, $13/2 = 7.5$. Although this technique is very simple, Howe used it to find dependencies between different events in the execution traces of complex computer programs, facilitating debugging (Howe and Cohen, 1995).

Finding Structure in Multiple Streams

We can also find dependencies between multitokens rather than just between token values in a single stream. For instance, the multitoken $< **R >$ precedes $< W** >$ more often than expected by chance under the hypothesis that multitokens are independent of their predecessors. Note the wildcards in the multitokens. The rule $< **R > \Rightarrow < W** >$ says, "when baby gazes at the rattle at time t, expect waving at time t+1." A more specific rule might include the state of the Tone stream; for example $< *HR > \Rightarrow < W** >$ says gazing at the rattle when its tone is high predicts waving. We have implemented two algorithms to find generalization hierarchies of such rules. The multi-stream dependency detection (MSDD) algorithm (Oates et al., 1995) starts with the most general possible rule (for the current example, $< *** > \Rightarrow < *** >$) and specializes it. The space of specializations is exponential, so MSDD uses a best-first search heuristic based on contingency tables for multitoken dependencies. For example, the rule $< **R > \Rightarrow < W** >$ and the streams above result in a contingency table (shown below in Figure 2) in which $< **R >$ is followed by $< W** >$ ten times, and by $\overline{< W** >}$ twice; whereas $\overline{< **R >}$ is followed by $< W** >$ twice and by $\overline{< W** >}$ seven times. This table is significant (G = 5.8, p=.017), but its specialization, for the rule $< *HR > \Rightarrow < W** >$, is not. Hence, MSDD doesn't explore this more specific rule until it has explored other, higher scoring rules. We have tested MSDD on artificially generated data sets and on standard classification problems from the UC Irvine collection. Note that classification tasks represent a subset of the problems to which MSDD can be applied. Classification typically involves finding a set of attribute values that accurately predict a single attribute value, whereas MSDD can find multiple attribute values that in turn accurately predict multiple attribute values. MSDD is surprisingly efficient and very accurate in comparison with other algorithms (Oates et al., 1995).

	$< W** >$	$\overline{< W** >}$	Totals
$< **R >$	10	2	12
$\overline{< **R >}$	2	7	9
Totals	12	9	21

Figure 2: Contingency table built from multitokens.

Incremental Multi-Stream Dependency Detection

MSDD is a batch algorithm, which means it needs to see all the streams before it starts work. That may not be possible or desirable in cases where learning about structure must be interleaved with acting on what has already been learned, or when structure in the environment changes over time. To overcome these problems, Matthew Schmill and Paul Cohen developed an incremental version of the MSDD algorithm, called IMSDD, that forms generalizations of multitoken prediction rules in a data driven manner.

IMSDD deals with precursor/successor pairs of multitokens, called words. As each new word appears, IMSDD parses the precursor portion through a data structure called a *precursor tree* by following the link at depth i that is labeled with the i^{th} token value in the precursor. Figure

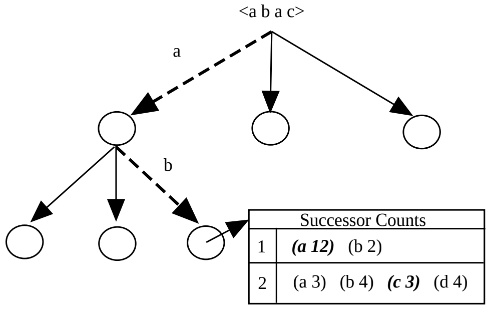

Figure 3: An example IMSDD precursor tree.

3 shows how $< AB >$, the precursor portion of $< ABAC >$, is parsed. Each leaf in the tree contains a successor table that indicates the frequency with which token values in particular streams follow the precursor that parsed to the leaf. Precursors can be generalized by looking for paths in the tree that differ in exactly one position and creating a new path with a wildcard in that position. For example, the precursors $< AB >$ and $< BB >$ would generalize to form the precursor $< *B >$. The successor tables of the more specific precursors are summed to initialize the successor table of the generalization. Given a precursor multitoken, a successor multitoken is predicted by heuristically selecting from among all successor tables reachable by parsing the precursor through the precursor tree (including paths with wildcards). To limit the size of the precursor tree, we employ a pruning method that dynamically excises old rules that have not proven to be accurate. Preliminary experiments with artificial data sets are encouraging (Schmill and Cohen, 1995).

Representing, Finding and Associating Ongoing Processes

Multitokens are a poor representation of ongoing processes; they are snapshots of the state of streams. In most real world environments, states persist for some amount of time before changing. In such situations, we are typically more interested in what the next state will be when the current one ends than what the state at time time $t + \delta$ will be given the state at time t. We have developed a representation for persistent states, called fluents, and an algorithm for finding fluents and dependencies among them.

The simplest kind of fluent is called a *base fluent*, and it represents a persistent token in one stream. For example, the fact that waving tends to continue once it has started is denoted W^+. We showed earlier how the G statistic captures the propensity of waving to follow waving, that is, to continue over time. The first step in fluent learning, then, is to discover base fluents by building contingency tables for individual token values in individual streams. When a token, say W, predicts itself significantly more often than one would expect by chance, we turn it into the base fluent W^+. Next in order of complexity are fluents we call *basic dependencies*. These have to do with temporal relationships between the beginnings and ends of fluents. For example, the `Open(Waving,Noise)` dependency is learned when the `Waving` fluent starts at or slightly before the time that the `Noise` fluent (a high tone followed by a low tone) starts. This happens, for example, when the baby is waving its rattle. Next come *composite fluents*. We currently form two kinds of composites: conjunctions and sequences. Composite fluents are formed by rules called fluent generators, of which we currently have two – one for conjuncts and one for sequences. The first says that if two fluents start together and end together, then they are the same thing, and conjoined. The second says that when an occurrence of one fluent predicts the occurrence of another, they are conjoined in a sequence. For example, the fluent shown below says that we expect the high tone (H^+) and the low tone (L^+) to follow each other in a sequence, and that they will occur in conjunction with waving (W^+).

```
(AND (SEQ H+ L+)
     W+)
```

As with base fluents, we use the G statistic to determine if observed co-occurrences of fluents represent true structure or are due to random chance. We have tested fluent learning in a simulated world in which a baby interacts with objects in its environment (Cohen et al., 1995). One of the goals of that project is to develop general algorithms that will allow an embedded agent to learn about the structure of its environment.

Conclusion

Single-stream dependency detection has helped us discover bugs in complex programs by examining their execution traces. Multi-stream dependency detection finds structure in artificially-generated streams, and performs very well on a related classification task with the Irvine Machine Learning datasets. IMSDD was developed to perform the same task as MSDD in an incremental manner. Fluent learning, in which we find evidence of ongoing processes and dependencies between them, has been tested in a complex artificial environment; we hope to test it with intensive care unit data (i.e., streams of readings from instruments over time) in the near future.

References

Cohen, P.R., Atkin, M., Oates, T. and D. Gregory. (1995) A representation and learning mechanisms for mental states. In *Working Notes of the AAAI Spring Symposium on Representing Mental States and Mechanisms.* pp. 15 – 21.

Howe, A.E. (1995) Finding Dependencies in Event Streams Using Local Search. In *Preliminary Papers of the Fifth International Workshop on AI and Statistics.* pp. 271 – 277.

Howe, A.E. and P.R. Cohen. (1995) Understanding Planner Behavior. To appear in *AI Journal.*

Oates, T., Gregory, D. and P.R. Cohen. (1995) Detecting Complex Dependencies in Categorical Data. In *Preliminary Papers of the Fifth International Workshop on AI and Statistics.* pp. 417 – 423.

Schmill, Matthew D. and P.R. Cohen. (1995) Learning Predictive Generalizations for Multiple Streams: An Incremental Algorithm. Department of Computer Science Technical Report 95-36, University of Massachusetts, Amherst.