

# Using Finite State Automata for Sequence Mining

Philip Hingston

School of Computer and Information Science  
Edith Cowan University  
2 Bradford St, Mt Lawley, WA 6050  
p.hingston@ecu.edu.au

## Abstract

We show how frequently occurring sequential patterns may be found from large datasets by first inducing a finite state automaton model describing the data, and then querying the model.

*Keywords:* data mining, sequence mining, automata, grammatical inference.

## 1 Introduction

Advances in computing have made it possible to capture and store huge collections of data. Examples of such collections include transaction databases, satellite images, DNA sequence databases and web site usage logs. The availability of these collections has created great interest in the problem of extracting useful knowledge from the data. In particular, much of this data is sequential in nature, so there is a need for techniques for exploring sequential patterns – what might be called sequence mining. In this paper, we show how a finite state automaton may be built that describes a sequential dataset, and how this model can then be used to provide answers efficiently to queries about frequencies of various patterns occurring in the data.

The structure of the paper is as follows. We first relate the sequence mining problem to the problem of regular grammatical inference. We then quickly review the relevant theory of finite state automata and regular languages, as it relates to grammatical induction. Next we derive formulae for calculating the frequencies of certain sequential patterns in a regular language, using a finite state automaton that generates the language. Finally, these are used to derive a sequence mining algorithm.

## 2 Sequence mining and grammars

The purpose of sequence mining is to discover useful sequential knowledge. Just what constitutes useful knowledge depends on the context – in market research it might be purchasing patterns (e.g. 40% of people who purchase a TV will later purchase a VCR); in web mining, it might be commonly occurring sequences of

web page hits. In any case, knowledge takes the form of insight into the structure of the data, since it is structure that makes things predictable, and it is predictability that can be exploited. In a recent article, Fayyad (2001) put it this way:

*“Accurate modeling of data is the ultimate form of data compression. If we derive the generative distribution describing a data set, we can reduce terabytes and petabytes of data to a single formula. These compact summaries of data become the ultimate portable data stores and can serve to answer queries and quickly navigate the data ...”*

We have applied this idea in the work described here – we seek a compact summary of a sequential data set in the form of a generative model (a grammar or automaton) and then use the model to answer queries about the data.

There are many existing techniques for finding structure in sequence data - Markov models, lag sequential analysis, and log-linear analysis to name a few. Recently, new methods especially suited to large datasets have been introduced – for example, Agrawal and Srikant (1995) have adapted algorithms for mining association rules to sequence mining. There is also the approach we have taken, to view the search for structure as a grammatical inference problem. Here is a small example to illustrate our approach. Suppose that we have a dataset with just four sequences, drawn at random from a larger collection:

$I = \{abb\text{aaaa}, ab, ba, \text{aaaaabb}\}$ .

Suppose also that the larger collection is a regular language, and can therefore be generated by a finite state automaton. Figure 1 shows a state graph for one possible generating automaton.

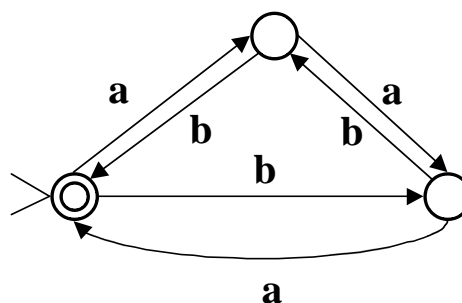


Figure 1: A Generating Automaton for I

In the figure, circles represent the states, and labeled arcs between them represent transitions. We follow the usual convention of marking the start state with a “>” and using

double-circles for the final states. Beginning at the start state, paths are traced through the graph by following transitions from one state to the next. The corresponding symbols are emitted as the transitions are traversed. Paths may stop at any final state. Assuming that the figure in fact shows the correct automaton, we can deduce pieces of knowledge about the larger collection. For example, we could deduce that the number of a's and the number of b's in any sequence are congruent modulo 3 (this property defines the larger collection in this case). If we were to assign probabilities to the transitions of the automaton, we could deduce, for example, the expected proportion of sequences starting with symbol a, or the proportion containing an a followed later by a b (we will show later how to make these calculations).

In the data mining setting, we usually seek patterns with a specified minimum degree of support (the number of confidence (the proportion of those sequences that match the pattern)). Support and confidence can be estimated using these calculated expected proportions.

Of course, some of these proportions could be “deduced” (really, estimated) by just counting the number of occurrences in the example data. But in sequence mining applications, the example sets in question may be very large, so counting may be a slow operation. We will see that the automaton and associated data structures are much smaller and the calculations required are efficient matrix operations. Another benefit of modeling the data with automata is that they are simple enough to be comprehended in an intuitive, visual way, suggesting regularities that would not be found by posing a standard set of statistical or other questions.

But before any calculation or deduction can be done, we must find an appropriate grammar or automaton to describe the data. We discuss how to do that in the next section.

### 3 Grammatical Inference

The problem of grammatical inference is to infer, from a set of example strings (and, possibly, non-examples) from a language, the set of rules, i.e. the grammar, which defines the language. In many cases, the target language is assumed to be a regular language. This is because regular languages are tractable, as well as being sufficiently powerful for most applications. A basic result of automata theory states that a language is regular if and only if it is generated by a finite state automaton. Thus, inferring a regular language is equivalent to finding a suitable automaton.

In the case of sequence mining, datasets normally have only examples, not non-examples, so we are interested in finding finite state automata that generate the given positive example data. We will now describe how such automata may be constructed.

For any finite set of sequences, there is a finite state automaton that generates (or equivalently accepts) exactly that set of sequences. One such automaton is the prefix tree acceptor (PTA) of the sequences. The PTA may be constructed by simply laying out the sequences, using a

state to represent each unique prefix of one of the sequences.

We can illustrate this using the sample data set I from above. The PTA of I is shown in the state diagram below, Figure 2.

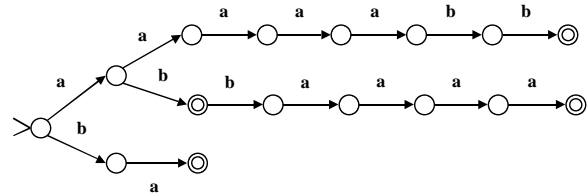


Figure 2: Example Prefix Tree Acceptor

A useful generalization of finite state automata is to add probabilities for the possible transitions from each state. We introduce a special delimiter symbol, which is not in the alphabet of the automaton, say “^”. If a state emits the delimiter symbol, this is taken to indicate the end of the string, and no next state is specified (or equivalently, the next state is understood to be the start state). Thus any state that can emit the delimiter symbol is considered a final state. We can now define a *probabilistic finite state automaton* as an automaton with transition probabilities, giving the probability that a particular symbol will be emitted when we are in a particular state. Figure 3 shows one possible set of transition probabilities for the automaton in Figure 2. Notice the “transitions” from the final states using the delimiter symbol.

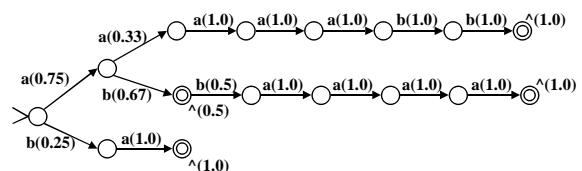


Figure 3: Automaton with Transition Probabilities

Other automata that generate the example data can be constructed from the PTA by merging together some of the states. In fact it can be shown that any automaton that generates the sample data and also satisfies additional “reasonableness” conditions can be obtained in this way (Dupont 1994). To illustrate this construction, consider what happens if we merge the two states that follow the start state in Figure 2. The resulting automaton would be non-deterministic, so further states must be merged to obtain a deterministic automaton, giving the automaton shown in Figure 4.

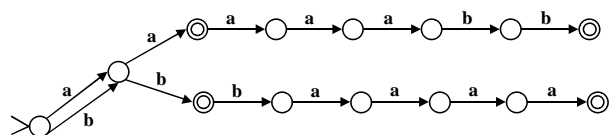


Figure 4: PTA after Some Merging

This automaton can generate the sequences of I, and also some sequences that begin with bb, for example, whereas the PTA only generates sequences beginning with aa, ab or ba. In general, merging states increases the set of sequences that can be generated. Many induction algorithms start with the PTA and choose states to merge so as to give the best description of the data (best in some sense that is particular to the algorithm).

When there are only positive examples, as would normally be the case for sequence mining, most algorithms use the statistical properties of the data to determine which states to merge, and the resultant models are stochastic – i.e. they describe the structure of the automaton as well as the transition probabilities. The Alergia algorithm (Carrasco 1994), for example, uses an information theory based test of similarity of state behaviours. Such algorithms can identify the target automaton with probability one, given sufficient data. They require many more examples than are needed for algorithms that use negative examples.

A different approach for positive examples only is to apply the principal of Occam’s razor, selecting the automaton that provides the “simplest” explanation of the data. This idea is behind a number of heuristic algorithms. The general pattern is to construct the PTA for the sample sequences, and then to search the space of sets of mergings, seeking to optimize a “figure of merit” or simplicity measure. These algorithms do not have the guaranteed convergence of Alergia, due to the incompleteness of the search, but they do not need as many examples. Algorithms of this kind are described in (Raman 1998) and (Hingston 2001).

#### 4 Estimating pattern frequencies from a finite state automaton

In this section, we give some examples to show how one can calculate the probabilities of particular patterns occurring in sequences generated by a probabilistic finite state automaton. These probabilities can then be used in sequence mining algorithms.

For our purposes, we will define a *pattern* using a sequence of n-grams of symbols. A sequence matches a pattern if the n-grams occur within the sequence in the correct order. We will use angle brackets to denote patterns. For example, the pattern <x, yz, w> matches any sequence containing an x, the digram yz, and w in that order, with or without intervening symbols. The empty pattern matches any sequence and is denoted by  $\lambda$ . If p1 and p2 are two patterns, p1 + p2 denotes the concatenation of the two patterns.

We need some notation for automata. If S is a state of an automaton and x is a non-terminal symbol on some transition from S, we will write

$p(S, x)$  = the probability that x is the symbol emitted,

$q(S, x)$  = the next state when x is emitted.

We will also use a particular automaton for illustration. We generated 100 sample sequences from the automaton

shown in Figure 1, randomly selecting transitions with equal probability. We then used an automaton inference algorithm to recover the original automaton structure from the sample data. The inferred automaton, shown in Figure 5, has transition probabilities determined from the data by counting transitions:

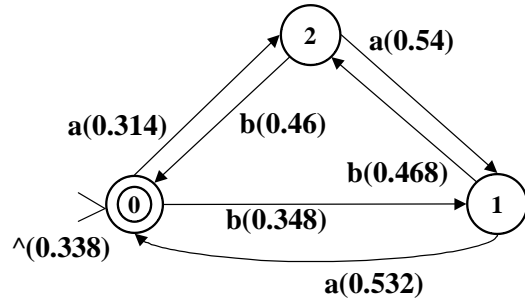


Figure 5: Inferred Automaton with Transition Probabilities

#### Example 1

“What proportion of sequences contain the symbol x?”

This question is perhaps the simplest kind of question we might ask. In terms of patterns, the question may be restated as asking for the probability of the pattern <x> occurring. We can compute this as follows. First we define, for each state S and non-delimiter symbol x:

$P(S, x)$  = the probability that a random path starting from state S contains an x.

Now a path starting from S and containing an x either begins with an x (with probability  $p(S, x)$ ), or begins with some other symbol and is followed by a path starting at the next state containing an x. Thus, we can write the formula:

$$P(S, x) = p(S, x) + \sum_{z \neq x} (p(S, z) \times P(q(S, z), x)) \dots (1)$$

In the case of the automaton in Figure 5, for symbol a, this formula gives:

$$P(0, a) = 0.314 + 0.348 \times P(2, a)$$

$$P(1, a) = 0.54 + 0.46 \times P(0, a)$$

$$P(2, a) = 0.532 + 0.468 \times P(1, a)$$

Thus we have a system of linear equations that can be solved to find  $P(S, x)$ . The answer to the original question is simply  $P(0, x)$ .

Equation (1) can be rewritten as follows:

$$P(S, x) = p(S, x) + \sum_{z \neq x} (p(S, z) \times P(q(S, z), x))$$

$$= p(S, x) + \sum_T \left( \sum_{\substack{z \neq x \\ q(S, z)=T}} p(S, z) \right) P(T, x) \dots (2)$$

We can write this in matrix form as follows.

First, define a matrix  $\rho(x)$  by

$$\rho_{S,T}(x) = \sum_{\substack{z \neq x \\ q(S,z)=T}} p(S,z)$$

For the automaton in Figure 5, we get

$$\rho(a) = \begin{bmatrix} 0 & 0.348 & 0 \\ 0 & 0 & 0.468 \\ 0.46 & 0 & 0 \end{bmatrix}$$

We write  $P(x)$  for the vector of values of  $P(S,x)$  and  $p(x)$  for the vector of values of  $p(S,x)$ . Then (2) becomes:

$$P(x) = p(x) + \rho(x)P(x)$$

Rearranging, we get

$$P(x) = (I - \rho(x))^{-1} p(x)$$

The matrix on the right hand side turns out to be a useful one, which we will denote by  $J(x)$ . Hence, we restate the formula above as

$$P(x) = J(x)p(x)$$

For the automaton in Figure 5, we have:

$$J(a) = \begin{bmatrix} 1.081 & 0.376 & 0.176 \\ 0.233 & 1.081 & 0.506 \\ 0.498 & 0.173 & 1.081 \end{bmatrix}$$

So that:

$$\begin{aligned} P(a) &= \begin{bmatrix} 1.081 & 0.376 & 0.176 \\ 0.233 & 1.081 & 0.506 \\ 0.498 & 0.173 & 1.081 \end{bmatrix} \times \begin{bmatrix} 0.314 \\ 0.54 \\ 0.532 \end{bmatrix} \\ &= \begin{bmatrix} 0.635 \\ 0.921 \\ 0.832 \end{bmatrix} \end{aligned}$$

Thus, in this case, the expected proportion of strings from the inferred automaton containing the symbol a is 0.635. The proportion obtained by counting sequences in the sample data is 0.63. The expected proportion from the original automaton in Figure 1, with equal probabilities for each symbol, is  $\overline{0.63}$ .

### Example 2

“What is the proportion of strings containing the symbol x, followed later by the symbol y?”

In terms of patterns, this may be restated as asking for the probability of the pattern  $\langle x, y \rangle$  occurring. To compute this we first define, for each pair of states S and T, and each non-delimiter symbol x:

$F(S,T,x)$  = the probability that a random path starting at state S and ending at state T contains exactly one x, which is the last symbol on the path.

Using similar reasoning as in Example 1, we see that

$$\begin{aligned} F(S,T,x) &= \sum_{z \neq x} (p(S,z) \times F(q(S,z),T,x)) + \\ &\begin{cases} p(S,x), & \text{if } q(S,x)=T \\ 0, & \text{otherwise} \end{cases} \dots (3) \end{aligned}$$

Once again we can rearrange this equation and write it in matrix form as follows.

Define a matrix  $\gamma(x)$  by

$$\gamma(S,T,x) = \begin{cases} p(S,x), & \text{if } q(S,x)=T \\ 0, & \text{otherwise} \end{cases}$$

Write  $F(x)$  for the matrix of values  $F(S,T,x)$ . Then equation (3) becomes:

$$F(x) = \gamma(x) + \rho(x)F(x)$$

As before, we deduce that:

$$F(x) = J(x)\gamma(x)$$

Returning to the original question, note that a sequence containing an x followed later by a y can be divided uniquely into one part that contains the first x in the sequence, and the following part, which contains a y. So if we write  $P(S, \langle x, y \rangle)$  = the probability that a random path starting at state S contains an x followed later by a y, we have:

$$P(S, \langle x, y \rangle) = \sum_T F(S,T,x)P(T,y)$$

Or in matrix form:

$$\begin{aligned} P(\langle x, y \rangle) &= F(x)P(y) \\ &= J(x)\gamma(x)J(y)p(y) \end{aligned}$$

In a similar fashion, we can compute:

$$\begin{aligned} P(\langle x, y, z \rangle) &= F(x)F(y)P(z) \\ &= J(x)\gamma(x)J(y)\gamma(y)J(z)p(z) \end{aligned}$$

and so on.

### Example 3

“What is the proportion of strings containing the digram  $xy$ ?”

If we define  $P(S, xy)$  = the probability that a random path starting at state  $S$  contains the digram  $xy$ , then in the same way as for Example 1 above, we can derive:

$$P(xy) = \tau(xy)J(x)$$

where

$$\tau(S, xy) = p(S, x)p(q(S, x), y)$$

By combining these techniques, we can derive formulae to compute probabilities for any desired ordering of symbols and  $n$ -grams. Other questions are also amenable to a similar analysis, e.g. “What is the expected length of sequences starting with  $x$ ?”

## 5 A sequence mining algorithm

In this section, we derive a sequence mining algorithm that uses the formulae from Section 4. We will search for rules like

*If a sequence contains an  $x$  followed by a  $y$ , then it is likely to contain a subsequent  $z$ .*

We will write this rule as  $\langle x, y \rangle \Rightarrow z$ .

More generally, we will search for patterns that tend to indicate that a particular symbol will follow. We will require that the rule have sufficient *support* (the probability that the pattern occurs) and *confidence* (the probability that the subsequent  $z$  occurs, given that the pattern has occurred). In the notation used in example 2, the support of the rule above is  $P(\langle x, y \rangle)$ , while the confidence is  $P(\langle x, y, z \rangle) / P(\langle x, y \rangle)$ . We can use the formula derived in Example 2 above to calculate probabilities of particular patterns occurring.

Suppose a support level,  $s$ , and a confidence level,  $c$ , have been specified. We define a *large* pattern as one having support of at least  $s$ . Large patterns have an Apriori property – that any prefix of a large pattern is also large – so we can use an Apriori-like scheme to make the search efficient.

---

### Algorithm Apriori-Pattern

---

Inputs:

A probabilistic finite state automaton  
support  
confidence

Output:

Set of rules  $R$

begin

$R = \{ \}$

$L_1 = \{ \}$

do (  $\forall s \in \Sigma$  )

compute  $F(s)$  and  $P(s)$  for the automaton

if  $P(\text{start}, s) \geq \text{confidence}$  then

$R = R \cup \{ \lambda \Rightarrow s \}$

if  $P(\text{start}, s) \geq \text{support}$  then

$L_1 = L_1 \cup \{ \langle s \rangle \}$

end do

$n = 1$

while  $L_n$  is not empty do begin

$L_{n+1} = \{ \}$

do (  $\forall \sigma \in L_n$  )

do (  $\forall s: \langle s \rangle \in L_1$  )

$\sigma' = \sigma + \langle s \rangle$

$P(\sigma') = F(\sigma)P(s)$

if  $P(\text{start}, \sigma') / P(\text{start}, \sigma) \geq \text{confidence}$  then

$R = R \cup \{ \sigma \Rightarrow s \}$

if  $P(\text{start}, \sigma') \geq \text{support}$  then

$L_{n+1} = L_{n+1} \cup \{ \sigma' \}$

end do

end do

$n = n + 1$

end while

end

---

## 6 Related Work

One of the most important applications for sequence mining is Web usage mining, where the data are sequences of Web page accesses extracted from Web site usage logs. Spiliopoulou et al have described a tool called Web Utilization Miner (WUM), which supports an SQL-like query language for sequences, including specification of a required level of support, and a template that candidate sequences are to match (Spiliopoulou 1999). These templates are somewhat similar to the example patterns we have explored. The mining algorithm creates an Aggregated Log, which is similar to the prefix tree acceptor, except that labels are on nodes rather than on arcs. Query results are given in the form of a graph formed by merging nodes from the matching portion of the Aggregated Log.

In (Agrawal 1995), Agrawal et al adapted the association mining algorithm, Apriori to find common subsequences in basket data. They present algorithms that provide efficient methods of finding all subsequences, or all maximal subsequences of such data with a given level of support. The term “subsequence” here allows intervening symbols, so is analogous to the idea of a template in WUM, or to the example patterns from the present paper. Several improvements have since been reported (Srikant 1996; Pei 2001).

In (Borges 1999), the authors describe a method of mining sequences from Web user navigation sequences. The method first uses the data to build a hypertext probabilistic grammar, which is a restricted form of

probabilistic regular grammar. Non-terminal symbols in the grammar (nodes in the transition graph) correspond to Web pages, and transitions correspond to links between pages. Our models are different in that symbols are associated with arcs rather than nodes and the resulting grammars are not so restricted.

## 7 Conclusion

We have presented a method for mining interesting sequential patterns from large sequential data sets. The first step is to model the data set in terms of a stochastic grammar or automaton. Queries about frequently occurring patterns in the data set can then be answered by converting pattern frequencies into formulae concerning the model, and using an Apriori-like scheme to enumerate the answers efficiently. In addition, the model provides a compact summary of the data set that aids understanding of its properties.

## 8 References

- AGRAWAL, R., & SRIKANT, R. (1995): Mining Sequential Patterns. *Proc. International Conference on Data Engineering (ICDE'95)*, Taipei, Taiwan, 3-14.
- BORGES, J., & LEVENE, M. (1999): Data Mining of User Navigation Sequences. *Proc. WebKDD'99*.
- CARRASCO, R.C., & ONCINA, J. (1994): Learning stochastic regular grammar by means of a state merging method. *Proc. The Second International Colloquium on Grammatical Inference (ICGI'94)*, Alicante, Spain, Lecture Notes in Artificial Intelligence, 139-152, Springer-Verlag.
- DUPONT, P., MICLET, L., & VIDAL, E. (1994): What is the search space of the regular inference? *Proc. Second International Colloquium on Grammatical Inference (ICGI'94)*, Alicante, Spain, Lecture Notes in Artificial Intelligence, 25-37, Springer-Verlag.
- FAYYAD, U. (2001): The Digital Physics of Data Mining. *Communications of the ACM* **44**(3):62-65.
- HINGSTON, P. (2001): Inference of Regular Languages using Model Simplicity. *Proc. Australian Computer Science Conference*, Gold Coast, Australian Computer Science Communications, **23**:8 pp, IEEE Press.
- PEI, J., HAN, J., MORTAZAVI-ASL, B., & PINTO, H. (2001): PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *Proc. 2001 Int. Conf. on Data Engineering (ICDE'01, Heidelberg, Germany)*.
- RAMAN, A., ANDREAE, P. & PATRICK, J. (1998): A Beam Search Algorithm for PFSA Inference. *Pattern Analysis and Applications* **1**:121-129.
- SPILIOPOULOU, M., FAULSTICH, L.C., & WINKLER, K. (1999): A Data Miner analyzing the Navigational Behaviour of Web Users. *Proc. Workshop on Machine Learning in User Modelling, ACAI'99*, Greece.
- SRIKANT, R., & AGRAWAL, R. (1996): Mining Sequential Patterns: Generalizations and Performance Improvements. *Proc. 5th International Conference on Extending Database Technology (EDBT'96)*, Avignon, France, 3-17.