Sequence Classification Using Statistical Pattern Recognition

José Antonio Iglesias, Agapito Ledezma, and Araceli Sanchis

Universidad Carlos III de Madrid,
Avda. de la Universidad, 30, 28911 Leganés (Madrid), Spain
{jiglesia,ledezma,masm}@inf.uc3m.es

Abstract. Sequence classification is a significant problem that arises in many different real-world applications. The purpose of a sequence classifier is to assign a class label to a given sequence. Also, to obtain the pattern that characterizes the sequence is usually very useful. In this paper, a technique to discover a pattern from a given sequence is presented followed by a general novel method to classify the sequence. This method considers mainly the dependencies among the neighbouring elements of a sequence. In order to evaluate this method, a UNIX command environment is presented, but the method is general enough to be applied to other environments.

Keywords: Sequence Classification, Sequence Learning, Statistical Pattern Recognition, Behavior Recognition.

1 Introduction

Sequential data mining is a broad discipline where the relationships of sequences of elements are used to different goals in different applications. A sequence is defined by the Merriam-Webster Dictionary as *a set of elements ordered so that they can be labelled with the positive integers*. Given a set of labelled training sequences, the main goal of a *sequence classifier* is to predict the class label for an unlabelled sequence. Furthermore, many other sequence learning tasks are considered: *sequence prediction* (given a set of sequences and one single sequence, predict the next item in the single sequence), *frequent subsequence discovery* (detect sub-sequences that occur frequent in a giving set of sequences), *sequence clustering* (cluster a set of unlabelled sequences in subsets), etc.

In particular, this paper focuses on the challenge of sequence classification. Let us define a sequence of n elements as $E = \{e_1, e_2, \dots, e_n\}$. Given a set of m classes $\mathrm{C} = \{c_1, c_2, \dots, c_m\}$ we wish to determine which class $c_i \in \mathrm{C}$ the sequence E belongs to. We present a novel method to classify a sequence.

This research is related to the framework used in the *RoboCup Coach Competition.* This competition of the Simulation League [1] was introduced in 2001, but changed recently in order to emphasize opponent-modelling approaches. The main goal of the current competition is to model the behavior of a team. A play pattern (way of playing soccer) is activated in a test team and the coach should detect this pattern and then, recognize the patterns followed by a team by observation.

We [2] presented a a very successful technique to compare agents behaviors based on learning the sequential coordinated behavior of teams. This technique was implemented by us in the 2006 Coach Team *Caos* [3].

In this paper, a sequence classifier is presented. As a sequence can represent an specific behavior, the classifier is evaluated in the environment of UNIX shell commands [4] in order to learn and classify a UNIX user profile.

The rest of the paper is organized as follows. In Section 2 we provide a brief overview of the related work on sequence classification. A summary of our approach is presented in section 3. Section 4 and 5 describe in detail the two parts of the proposed technique: Pattern Extraction and Classification. Experimental results are given in section 6. Finally, section 7 contains future work and concluding remarks.

2 Related Work on Sequence Classification

The main reason to need to handle sequential data is because of the observed data from some environments are inherently sequential.

An example of these environments is the DNA sequence. Ma et al. [5] present new techniques for bio-sequence classification. Given an unlabelled DNA sequence S, the goal in that research is to determine whether or not S is an specific promoter (a gene sequence that activates transcription). Also, a tool for DNA sequence classification is developed by Chirn et al. [6].

In the computer intrusion detection problem, Coull et al. [7] propose an algorithm that uses pair-wise sequence alignment to characterize similarity between sequences of commands. The algorithm produces an effective metric for distinguishing a legitimate user from a masquerader. In [8] Schonlau et al. investigate a number of statistical approaches for detecting masqueraders.

Another important reason to research sequential data is its motivation in the domain of user modelling. Bauer [9] present an approach towards the acquisition of plan decompositions from logged action sequences. In addition, Bauer [10] introduces a clustering algorithm that allows groups of *similar* sequences to be discovered and used for the generation of plan libraries.

In the area of agent modelling, Kaminka et al. [11] focus on the challenge of the unsupervised autonomous learning of the sequential behaviors of agents, from observations of their behavior. Their technique translates observations of a complex and continuous multi-variate world state into a time-series of recognized atomic behaviors. These time-series are then analyzed to find sub-sequences characterizing each agent behavior. In this same area, Riley and Veloso [12] present an approach to do adaptation which relies on classification of the current adversary into predefined adversary classes. This classification is implemented in the domain of simulated robotic soccer.

In Horman and Kaminka's work [13] a learner is presented with unlabelled sequential data, and must discover sequential patterns that characterize the data. Also, two popular approaches to such learning are evaluated: frequency-based methods [14] and statistical dependence methods [4].

3 Our Approach

In this work, the input consists of a set of sequences. A sequence is an ordered list of elements (events, commands,...) that represents an specific behavior (pattern). In the proposed framework, each sequence designates a class. The first part of this classifier is to discover and store the pattern (class) followed by each sequence. Then, a new small sequence is observed and compared to the stored patterns (classes) in order to determine which class it belongs to.

Therefore, the proposed approach has two main phases (Figure 1 shows an overview structure):

1. **Pattern Extraction.** A pattern can be defined as a compact and semantically sound representation of raw data (sequence). In our approach, every input sequence follows a different pattern, so a sequence pattern represents a class. Every sequence is pre-processed and represented in a special structure in order to get the pattern that it follows. This phase creates a library where the patterns obtained from each sequence are stored.
2. **Classification.** Once every pattern has been stored, a given sequence must be classified. The pattern of the given sequence is generated using the pattern extraction process. This pattern is then matched to every pattern in the *Our Patterns Library.*

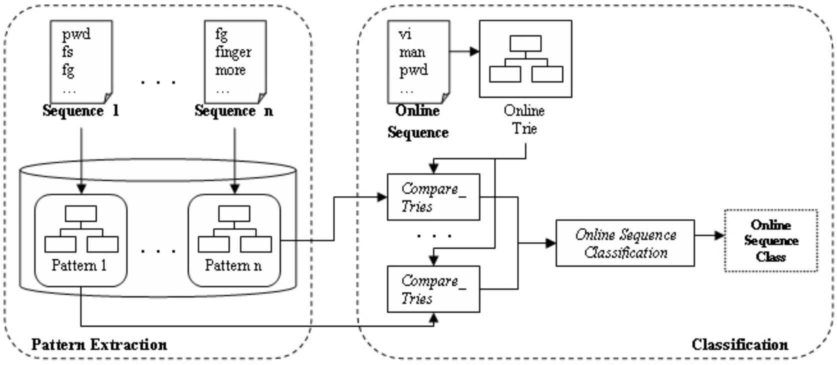

Fig. 1. Overview structure

4 Pattern Extraction

A sequence is an ordered list of elements that follows a pattern and it can be represented as $\{e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_n\}$ where n is the length of the sequence. As a pattern represents a compact and semantically sound representation of the sequence, the first step is to extract from the sequence the elements more related to it. Also, a pattern should be predictable, so we consider that the repeating elements of the sequences and its dependencies are related to the pattern.

Because of the previous supposition, in this work we propose the use of a trie data structure [15] [16] to store the useful sequence information. Therefore, the output of this first phase is a library in which the trie of each sequence is stored. As a trie represents the pattern followed by a sequence, this library is called *Pattern Library*.

4.1 Building a *Trie*

A trie (abbreviated from *retrieval*) is a kind of search tree similar to the data structure commonly used for page tables in virtual memory systems. This special search tree is used for storing strings in which there is one node for every common prefix and the strings are stored in extra leaf nodes.

The trie data structure has been used for retrieving a string efficiently from a set of strings; in [11] is used to learn a team behavior and in [17] to create frequent patterns in dynamic scenes. In this research we propose to use this data structure for a different goal: to store the main characteristics of a sequences in an effective way. The advantage of this kind of data structure is that every element is stored in the trie just once, in a way that each element has a number that indicates how many times it has been inserted on.

In the proposed trie structure, every element of the sequence is represented as a node. A path from the root to a node represents an ordered list of elements. Also, as the length of the sequences could be very long, the sequence must be split into smaller sub-sequences in order to store its elements in a trie. The length of these sub-sequences can modify both the size of the tries and the final results quite significantly.

The size of a trie depends on both the inserted and the repeated nodes. Due to the repeated nodes can vary with the sequence to treat; to analyze the relation between the sub-sequence length and the size of the generated trie, the number of nodes to insert is measured. Figure 2 shows the correlation between the sub-sequence length of a 100 elements sequence and the number of nodes (elements) to insert in the trie. As shown in Figure 2, given a sequence of n elements, to increment in one unit the length of the sub-sequence results in inserting $n/2$ elements in the trie. Therefore, the sub-sequence length is crucial in the proposed method.

Steps of Creating an Example *Trie*. An example of how to store a sequence in a trie data structure is shown as follows. In this example, a sequence consists of different words, which represent any kind of element. The sequence to insert into an initially empty trie is:

$\{w5 \rightarrow w1 \rightarrow w5 \rightarrow w1 \rightarrow w5 \rightarrow w3\}$

Firstly, this sequence must be split. Let 3 be the sub-sequence length, then the sequence is split in two sequences:

$\{w5 \rightarrow w1 \rightarrow w5\}$ and $\{w1 \rightarrow w5 \rightarrow w3\}$

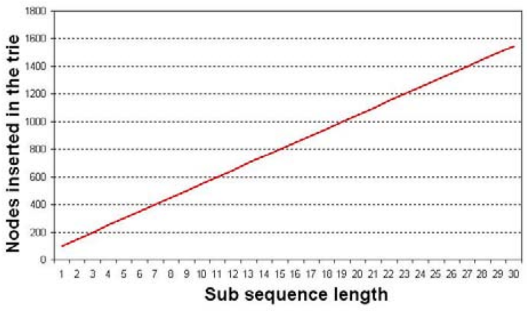

Fig. 2. Correlation between the sub-sequence length of a 100 elements sequence and the number of elements to insert in the trie

The first sequence is added as the first branch of the trie (Figure 3 A). Each element is labelled with the number 1 that indicates that the element has been inserted in the node once (in Figure 3, this number is enclosed in brackets). Because of repeating and significant sub-sequences are important to find the sequence pattern, the suffixes of the sub-sequences are also inserted. In the example, the suffixes $\{w1 \rightarrow w5\}$ and $\{w5\}$ are then added to the trie (Figure 3 B). Finally, after inserting the second sub-sequence and its remaining suffix, the complete trie is obtained (Figure 3 C).

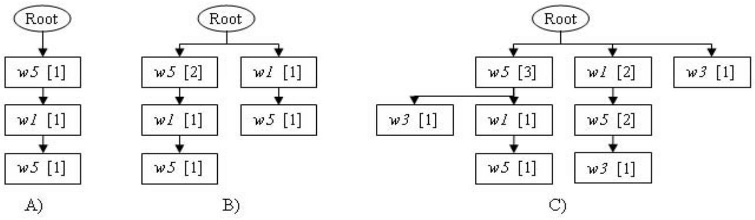

Fig. 3. Steps of creating an example trie

4.2 Evaluating Dependencies

In order to find the pattern that characterizes the elements of the sequence stored in a trie, two different approaches can be considered: frequency-based methods [14] and statistical dependence methods [4]. Considering the experimental results in [13], in this research, a statistical dependence method is used. In particular, to evaluate the relation between an element and its prefix (succession of elements previous to an element), we use one of the most popular statistic methods: the Chi-square test [18]. This statistical test enables to compare observed and expected element sequences objectively and evaluate whether a deviation appears. Hence, every element (node) of a trie stores a value that determines whether an element is or not relevant with the previous ones.

To compute this test, it is necessary a 2x2 contingency table (also known as a cross-tabulation table). This table is filled with four frequency counters, as shown in Table 1. The counters are calculated as follows: The first number O_{11} indicates how many times the current element (node) is following its prefix. The number O_{12} indicates how many times the same prefix is followed by a different element. The number O_{21} indicates how many times a different prefix of the same length, is followed by the same element. The number O_{22} indicates how many times a different prefix of the same size, is followed by a different element.

Table 1. Contingency table

	Element	Different element	Total
Prefix	O_{11}	O_{12}	$O_{11} + O_{12}$
Different prefix	O_{21}	O_{22}	$O_{21} + O_{22}$
Total	$O_{11} + O_{21}$	$O_{12} + O_{22}$	$O_{11} + O_{12} + O_{21} + O_{22}$

The expected values are calculated as in Equation 1

$$Expected(E_{ij}) = \frac{(Row_i TotalxColumn_i Total)}{GrandTotal} \tag{1}$$

The formula to calculate chi-squared value, is given in equation 2.

$$X^2 = \sum_{i=1}^{r} \sum_{j=1}^{k} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{2}$$

where: O_{ij} is the observed frequency and E_{ij} is the expected frequency.

This value is calculated for each element of the trie. Hence, the trie structure obtained in section 4.1. is modified to include this value in every node.

Finally, and as result of the first phase, every created trie (that represents a sequence pattern), is stored in the *Pattern Library.*

5 Classification

Given a new (and usually small) sequence to classify, the goal of this process is to determine which pattern (from the *Pattern Library*) the sequence is following. This process compares the given sequence to every pattern stored in the library. Therefore, the first step is to create the trie (that represents the pattern) corresponding to the sequence to classify. This trie (which we call *Testing Trie*) is generated using the process explained in section 4. This *Testing Trie* is then compared to every trie of the *Pattern Library.* Before describing the comparing algorithm, we should remember that every node in a trie is represented by: *Element* (word that indicates a specific element), *Prefix* (set of previous elements in the trie branch) and *Chi-Sq* (number that indicates the chi-square value for the node).

5.1 *Trie Sub-comparison*

If the *Testing Trie* and a trie from the *Pattern Library* (which we call *Class Trie*) represent the same pattern; the recurring elements to recurring prefixes should be similar in both tries. Accordingly, the key to compare two different tries is to note the possibility (measured by the chi-square value, *chi-sq*) that an element (e) occurs after a prefix (p) in both tries. In our method, the similarities and differences of two tries are represented by a set of *Trie sub-comparison* data structure, which can be defined as follow: *Trie sub-comparison* = (e, p, *comparison Value*) where the *comparison Value* represents the similarity or difference in both tries regarding the element e and its prefix p. This value is calculated from its chi-square values.

5.2 The Comparing Algorithm

The inputs of the algorithm presented below are the two tries to be compared. To apply this algorithm for a classifier method, it is executed once for every trie stored in *Pattern Library*. The number of executions is the number of classes (tries in library) and the two inputs are: the *Testing Trie* and a *Class Trie*.

The main points of the proposed comparing algorithm are the following:

For each node of the *Testing Trie*, its element and prefix are obtained. In the *Class Trie*, then a node with the same element and prefix is sought:

- If the present node is only in the *Testing Trie*:
  - It is interpreted as a difference between both tries. This difference together with the element and its prefix, are stored as part of the comparing result in the proposed structure *Trie sub-Comparison*. In this structure, the *comparison Value* indicates that there exists a difference between both tries. The *comparison Value* is the chi-square of the present node but its value is stored as a minus value because is representing a difference. As higher is the chi-square value, as more representative is the difference.
- If a node with the same element and prefix is in both tries:
  - The Chi-Square value of both nodes is compared: If the difference is lower than a threshold value, it means that there is some kind of similarity between the two tries. In this case, the *comparison Value* is the chi-square of the present node but it is stored as a positive value because is representing a similarity.

Figure 4 presents the basic structure of the proposed algorithm. The result of the algorithm is a set of *Trie Sub-Comparison* (*Comparison-Result*) that describes the similarities and differences of both tries. In this algorithm are used the following functions: *depthTrie(Trie T)*: returns the maximum depth of any of the leaves of the trie T. *getSetOfNodes(Level L, Trie T)*: returns a set nodes of the trie T in the level L. *getNode (Element E, Prefix P, SetOfNodes S)*: returns a node (from the set of nodes S) consisting of the element E and which prefix is P. (If a node with these parameters does not exist in S, the function returns

Algorithm 1. CompareSimilarityTries (*TestingTrie, ClassTrie*)

```
for $level_i \leftarrow 2$ to depthTrie ($TestingTrie$) do
    $set_t \leftarrow$ getSetOfNodes($level_i$, $TestingTrie$)
    $set_c \leftarrow$ getSetOfNodes($level_i$, $ClassTrie$)
    for all $node_t$ in $set_t$ do
        $node_c \leftarrow$ getNode (element($node_t$), prefix($node_t$), $set_c$)
        if ($node_c$ == null)   {the node is only in the Testing Trie}
            Trie-sub-Comparison $\leftarrow$ Add(element($node_t$), prefix($node_t$), chi-sq ($node_t$)*-1)
            Comparison Result $\leftarrow$ Add(Trie-sub-Comparison)
        else   {The node is in both tries}
        if (abs(chi-sq($node_t$) - chi-sq ($node_c$)) $\leq$ ThresholdValue)
            Trie-sub-Comparison $\leftarrow$ Add(element($node_t$), prefix ($node_t$), chi-sq ($node_t$))
            Comparison-Result $\leftarrow$ Add(Trie-sub-Comparison)
        end if
    end for
end for
```

Fig. 4. Basic Structure of the Comparing Algorithm of two tries

null). Finally, *element(node N)*, *prefix(node N)* and *chi-sq(node N)*: return the element, prefix and chi-square of the node *N*, respectively.

Once the *Testing Trie* has been compared with every *Class Trie*; we add up the *comparison Value* for every *Trie Sub-Comparison* obtaining an amount for each *Class Trie*. This amount represents the similarity between the given sequence and the class. Therefore, the result of the classifier is the *Class Trie* with a higher value (positive values represent similarity).

6 Experimental Setups and Results

In order to evaluate the proposed method, we have fully implemented a system that classifies UNIX command line sequences. In this environment, we extract the profile of a user from its UNIX commands sequences and then we classify a given sequence in one of the user profile previously stored. This task is very useful in computer intrusion detection.

We used 9 sets of sanitized user data drawn from the command histories of 8 UNIX computer users at Purdue University over 2 years [19]. The data is drawn from *tcsh* history files and has been parsed and sanitized to remove filenames, user names, directory structures, etc. Command names, flags, and shell metacharacters have been preserved. Additionally, tokens have been inserted to divide different user sessions. Also, and it is very important in our research, tokens appear order issued within the shell session, but no timestamps are included in the data.

For evaluating the proposed method, we have used the benchmark data sets from [19]. Each input file contains from about 10.000 to 60.000 commands. Firstly, for each user is created a trie (user profile library) that represents its

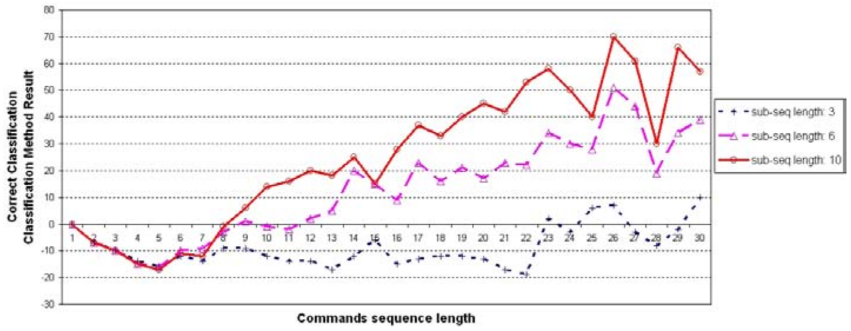

Fig. 5. Comparing Results. Unix Commands Classification - User 6.

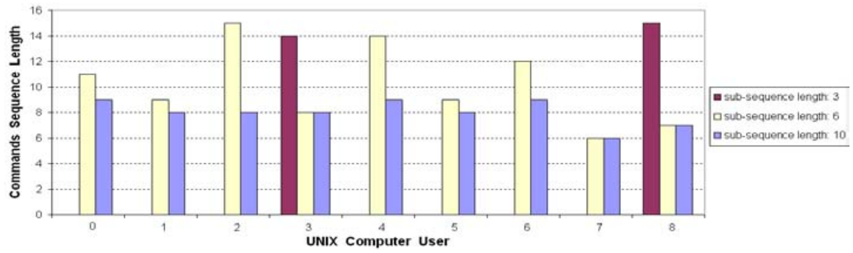

Fig. 6. Length necessary to classify a UNIX computer user correctly

behavior. As we explained in section 4, the length of the sub-sequences can modify both the size of the tries and the final results quite significantly, so we have executed our method with different sub-sequences length in order to evaluate the results.

Once tries of different sizes have been built for each user, we conducted extensive experiments. To evaluate our method we use the user profile library (set of classes) and a given sequence to be classified (this sequence is labelled because it is obtained from a user file). As we want to recognize a user as soon as possible, we classify sequences of very different sizes. After using the proposed technique, a comparing value is obtained for each user profile (*class value*). From these results, the given sequence is classified in the class with the highest value.

In order to evaluate the result and represent them graphically, for each given sequence we calculate a result value. This value represents the difference between the value obtained for the given sequence class and the highest *class value*:

– If the obtained value is negative, it means that there is another class considered by our method more similar to the given sequence (our classification is wrong).

– If the value is zero, the classification is right. Also, to evaluate the correctness of the result and the efficiency of the algorithm, the following value is calculated: difference between the obtained value and the second highest value. Therefore, as higher is this calculated value, as better is the classification.

Figure 5 shows the results for a sequence obtained from the user 6 commands file. The X-axis represents the given sequence length. The Y-axis represents the calculated value to evaluate our method. In the graph, three different sub-sequence lengths (3, 6 and 10) are represented. Because of the result of the method can depend on the given sequence, each point represented in the graph is the average value of 25 different tests conducted. As we can see, the best result is obtained using long sub-sequences. However, the size of the trie and the time consuming to build the trie and classify the sequence are highly increased with the length of the sub-sequences.

Because of lack of space, we have omitted the graphs that represent the result for the other 8 users. However, these results are also successful and the representative values are similar. Considering the results, we obtain: Let 6 be the sub-sequence length, then this method is able to correctly classify every given sequence of more than 15 commands.

Figure 6 represents the length of the given sequence necessary to classify correctly one of the 9 evaluated users. Considering a sub-sequence length of 3, the classification is not usually correct after even 50 commands. Only two users (3 and 8) are correctly classified with this size.

7 Conclusions and Future Work

In Horman and Kaminka's work [13] a learner to discover sequential patterns is presented. Also, to overcome the length bias obstacle, they normalize candidate pattern ranks based on their length. To improve the results in our research, a normalization method for comparing tries of different lengths could be implemented.

Previous to this research, we have developed a method for comparing agents behavior. The method was based on learning the sequential coordinated behavior of teams. The result of that method was successfully evaluated in the *RoboCup Coach Competition*. Related to that research, in this paper a sequence classification using statistical pattern recognition is presented. This method consists of two different phases: Pattern extraction and Classification. The goal of the first phase (in which previous works have been considered) is to extract a pattern or behavior from a sequence. The extracted pattern is represented in a special structure: *trie*. The second phase presents a method to compare different patterns (*tries*) in order to classify a given sequence.

In order to evaluate the proposed technique in a specific environment, we focus our experiments on the task of classify UNIX command line sequences. The technique was evaluated in a rigorous set of experiments and the results demonstrate that it is very effective in such tasks.

On the other hand, other approaches have been applied in the environment presented in this paper (UNIX shell commands): using *Hidden Markov Models* (HMMs) [20] [21] or employing instance-based learning (IBL) [20]. However, the goals to achieve by these methods differ from our proposal. A detailed analysis confronting our approach with others is proposed as future work.

Acknowledgments. This work has been supported by the Spanish Government under project TRA2004-07441-C03-2/IA.

References

1. The robocup 2005 coach competition web page (December 2006), `http://staff.science.uva.nl/~jellekok/robocup/rc05`
2. Iglesias, J.A., Ledezma, A., Sanchis, A.: A comparing method of two team behaviours in the simulation coach competition. In: Torra, V., Narukawa, Y., Valls, A., Domingo-Ferrer, J. (eds.) MDAI 2006. LNCS (LNAI), vol. 3885, pp. 117–128. Springer, Heidelberg (2006)
3. Iglesias, J.A., Ledezma, A., Sanchis, A.: Caos online coach 2006 team description. In: CD RoboCup 2006, Bremen, Germany (2006)
4. Howe, A.E., Cohen, P.R.: Understanding planner behavior. Artificial Intelligence 76(1-2), 125–166 (1995)
5. Ma, Q., Wang, J.T.-L., Shasha, D., Wu, C.H.: Dna sequence classification via an expectation maximization algorithm and neural networks: a case study. IEEE Transactions on Systems, Man, and Cybernetics, Part C 31(4), 468–475 (2001)
6. Chirn, G.-W., Wang, J.T.-L., Wang, Z.: Scientific data classification: A case study. ICTAI '97: Proceedings of the 9th International Conference on Tools with Artificial Intelligence , 216 (1997)
7. Coull, S.E., Branch, J.W., Szymanski, B.K., Breimer, E.: Intrusion detection: A bioinformatics approach. In: Omondi, A.R., Sedukhin, S. (eds.) ACSAC 2003. LNCS, vol. 2823, pp. 24–33. Springer, Heidelberg (2003)
8. Schonlau, M., DuMouchel, W., Ju, W., Karr, A., Theus, M., Vardi, Y.: Computer intrusion: Detecting masquerades, statistical Science (2001) (submitted)
9. Bauer, M.: Towards the automatic acquisition of plan libraries. ECAI , 484–488 (1998)
10. Bauer, M.: From interaction data to plan libraries: A clustering approach. In: IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pp. 962–967. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)
11. Kaminka, G.A., Fidanboylu, M., Chang, A., Veloso, M.M.: Learning the sequential coordinated behavior of teams from observations. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002. LNCS (LNAI), vol. 2752, pp. 111–125. Springer, Heidelberg (2003)
12. Riley, P., Veloso, M.M.: On behavior classification in adversarial environments. In: Parker, L.E., Bekey, G.A., Barhen, J. (eds.) DARS, pp. 371–380. Springer, Heidelberg (2000)
13. Horman, Y., Kaminka, G.A.: Removing statistical biases in unsupervised sequence learning. In: IDA, 2005, pp. 157–167 (2005)
14. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Yu, P.S., Chen, A.S.P. (eds.) Eleventh International Conference on Data Engineering, pp. 3–14. IEEE Computer Society Press, Taipei, Taiwan (1995)

15. Fredkin, E.: Trie memory. Comm. A.C.M. 3(9), 490–499 (1960)
16. Knuth,: The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1973)
17. Huang, Z., Yang, Y., Chen, X.: An approach to plan recognition and retrieval for multi-agent systems. In: Proceedings of AORC (2003)
18. Chiang, C.L.: Statistical Methods of Analysis, World Scientific, Suite 202, 1050 Main Street, River Edge, NJ 07661 (2003)
19. Newman, C.B.D.J., Hettich, S., Merz, C.: UCI repository of machine learning databases (1998), `http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html`
20. Lane, T., Brodley, C.E.: An empirical study of two approaches to sequence learning for anomaly detection. Mach. Learn. 51(1), 73–107 (2003)
21. Yeung, D.-Y., Ding, Y.: Host-based intrusion detection using dynamic and static behavioral models. Pattern Recognition 36(1), 229–243 (2003)