Improving Sequence Learning for Modeling Other Agents

Yoav Horman and Gal A. Kaminka
Department of Computer Science
Bar-Ilan University, Israel

Abstract

To accomplish in their tasks, agents need to build models of other agents from observations. In open or adversarial settings, the observer agent does not know the full behavior repertoire of observed agents, and must learn a model of the other agents from its observations of their actions. This paper focuses on learning models of sequential behavior *based on observed execution traces. It empirically compares sequence recognition approaches, and shows that they suffer from common deficiencies, including length-biases and inability to generalize discovered patterns. We present bias-removing and clustering methods to address these challenges, and evaluate them using synthetic and real-world data. The results show significant improvements in all the learning algorithms tested.*

1. Introduction

To accomplish in their tasks, agents need to build models of other agents from observations. In open or adversarial settings, the observer agent does not know the full behavior repertoire of observed agents, and must learn a model of the other agents from its observations of their actions. Examples of such settings include monitored human-computer interactions [2], or coaching a robotic soccer team, a task where a synthetic coach must determine tactics to counter opponent teams [7].

We focus in this paper on learning a model of the sequential behavior of agents by analyzing their observed actions, *without supervision*. Two general approaches to this task are frequency-based methods [1, 12], and statistical dependence methods [5, 6]. However, to the best of our knowledge, their relative strengths with respect to sequence learning have not been evaluated. Furthermore, common challenges or merits have not been identified.

Our first contribution is in carrying out a comparative evaluation of the above methods. Based on a unifying underlying representation using an augmented suffix trie [9], we apply these methods in several domains, and compare their performance. The results show that statistical dependence techniques typically fare significantly better than frequency-based ones. However, more importantly, the comparison uncovers several common deficiencies to the methods we tested. In particular, we show that they are (i) biased in preferring sequences based on their length; and (ii) may incorrectly differentiate between similar sequences that reflect the same general pattern.

The paper tackles these deficiencies. First, we show that a length normalization method leads in fact to significant improvements in all learning methods (up to 42%). We then show how to use clustering to group together similar sequences. We show that previously distinguished subpatterns are now correctly treated as instances of the same general pattern. This results in additional significant precision improvements. We extensively evaluate these techniques using synthetic data and on two real data sets. The experiments show that the techniques are generic, in that they significantly improve all of the methods initially tested.

2. Background: Sequence Recognition

The literature reports on several alternative unsupervised sequence learning techniques. Frequency based algorithms pick the patterns whose frequency in the segments passes a user-specified threshold—called *minimal support*—specified as percentage of the input segments [1]. Unfortunately, support-based methods face difficulty ignoring patterns that are due to chance—patterns that emerge from the likely frequent co-occurrence of a frequent suffix and a frequent prefix. To address this difficulty, support-based recognition is usually coupled with another technique, *confidence*, which measures the predictability of a pattern suffix given its prefix. In such combinations, the extracted patterns are those that are more frequent than the user-specified *support*, and more predictive than the user-specified *confidence*.

Even taking confidence into account, support-based methods still provide misleading results [11] and cannot recognize highly predictive yet rare patterns[7]. For example, in UNIX command-line sequences, the pattern $edit \rightarrow make \rightarrow a.out$ (implying a programmer's EDIT-COMPILE-DEBUG cycle) is likely to be more frequent than expected, given the individual frequency of its constituent events.

	α_k	$\neg\alpha_k$	
p_r	n_1	n_2	$freq(p_r)$
$\neg p_r$	n_3	n_4	$\sum_{p_m \neq p_r} freq(p_m)$
	$freq(\alpha_k)$	$\sum_{\alpha_i \neq \alpha_k} freq(\alpha_i)$	

Table 1: A statistical contingency table for sequential pattern p, composed of a prefix $p_r = \alpha_1, \alpha_2, \ldots, \alpha_{k-1}$ and suffix α_k.

Dependency-detection algorithms can recognize statistically significant patterns ([5, 6]). These test for the statistical dependence of a suffix on its prefix, taking into account the frequency of other prefixes and suffixes. To calculate the rank of a given pattern p, a 2×2 contingency table is built for its prefix p_r and suffix α_k (Table 1). In the top row, n_1 is the number of times that we saw the pattern p (p_r followed by α_k), and is simply $freq(p)$. n_2 is the number of times we saw a different suffix to the same prefix, i.e., $\sum_{\alpha_i \neq \alpha_k} freq(p_r \alpha_i)$. In the second row, n_3 is the number of patterns in which α_k followed a different prefix than p_r ($\sum_{p_m \neq p_r} freq(p_m \alpha_k)$). n_4 is the number of patterns in which a different prefix was followed by a different suffix ($\sum_{p_m \neq p_r} \sum_{\alpha_i \neq \alpha_k} freq(p_m \alpha_i)$). The table margins are the sums of their respective rows or columns. A chi-square or G test [5] is then run on the contingency table to calculate the dependence of α_k on p_r.

3. Recognition: A Comparison

To the best of our knowledge, a comparison contrasting the relative performance of the different approaches discussed above has not been done. To carry out such a comparison, we use a representation which stores all subsequences in such a manner that allows calculating their ranks via different methods.

Our representation is composed of two data structures [7]. The first is an augmented suffix trie [9], a tree-like data structure in which each node represents an event, and its children represent events that have immediately followed it. A sequence is represented by a path from the trie's root to the node that contains the last event of the sequence. Each sequence is inserted to the trie together with all its suffixes. Each node in the trie keeps track of the number of times it has been passed through. Contingency table calculation requires an additional data structure, which maintains summary information per each level in the trie. For each level l, and event e, we maintain (i) $ecount(e, l)$, the number of times event e was inserted at level l; and (ii) $lcount(l)$, the number of all sequences of length l. Using this information we can calculate support, confidence and dependency ranks for of all the sequences in the database. It is then straightforward to extract the top subsequences according to each method, by finding the k best nodes.

We conducted extensive recognition experiments using synthetic data, comparing support, confidence, support/confidence and dependency-detection (hereinafter, marked *DD*). In each run, the techniques above were to recognize five different re-occurring *true patterns*, uniformly distributed within a file of 5000 segments. We refer to the percentage of these segments that contain actual patterns as *pattern rate*. Such segments may include additional random events before or after the true pattern. We also controlled *intra-pattern noise rate*: the probability of having noise events within a pattern.

In each experiment, each technique reported its best 10 pattern candidates, and those were compared to the five true patterns. The results were measured as the percentage of true patterns that were correctly detected. The support/confidence technique requires setting manual thresholds. To allow this method to compete, we set its thresholds such that no true pattern would be pruned prematurely. We refer to this technique as "Support/Confidence Optimal". We have also tested a more realistic version of the algorithm, using fixed minimal confidence of 20% ("Support/Confidence").

For three different values of alphabet size T (5, 10 and 26) and three ranges of true-pattern sizes (2–3, 3–5, 4–7) we have generated data sets of sequences with incrementing pattern rates. Intra-pattern-noise was fixed at 0%. For each pattern rate we have conducted 50 different tests. Overall, we ran a total of 4500 tests, each using different 5000 sequences and a different set of 5 true patterns.

The results are depicted in Figure 1. In the figure, the x-axis measures the pattern rate from 0.2% to 100%. The y-axis measures the average accuracy of the different techniques over the various combinations of T and pattern size. The figure shows that dependency-detection (*DD*) outperformed other methods for low and medium values of pattern rate. The optimal support/confidence algorithm usually came second, outperforming DD at higher pattern rates. However, the standard support/confidence, as well as the simplistic support technique, have provided poor results.

Figure 2 shows the results for the same experiment, focusing on pattern rates lower than 5%. As can be clearly seen, DD quickly achieves relatively high accuracy, significantly higher than the support/confidence method.

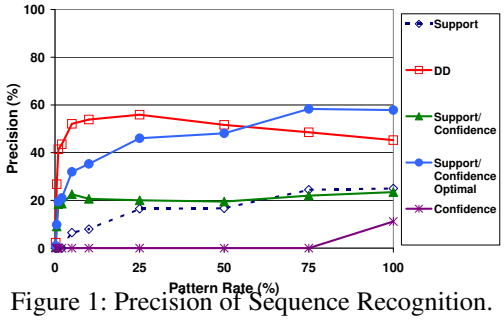

Figure 1: Precision of Sequence Recognition.

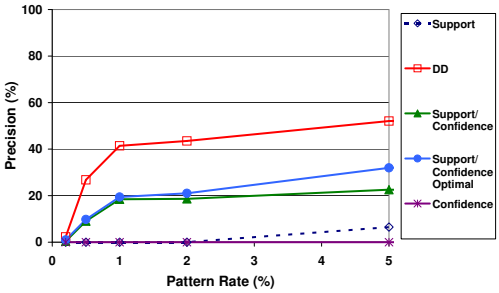

Figure 2: Precision at Low Pattern Rates.

4. Improving Recognition

We analyze the failures and successes of the different techniques. DD was better than support-confidence, but did not necessarily fair well on an absolute scale. Support-based methods did not appropriately handle low pattern rates. But more importantly, we have found that all methods suffer from common limitations.

4.1. Removing Length Bias

The first significant limitation of the approaches we describe above is their bias with respect to the length of generated patterns. Support-based methods are naturally biased towards short sequences, which are much more frequent than others. DD methods are also susceptible to this bias since they take into account the absolute frequency involved when assessing the significance of the result. Shorter sequences are more frequent and on an absolute scale typically receive more confident dependency results.

In order to overcome this obstacle, we normalize candidate pattern ranks based on their length. This method, used in [3] for unsupervised segmentation of observation streams based on statistical dependence tests, is also useful in support and confidence techniques.

The key to this method is to normalize all ranking based on units of standard deviation, which can be computed for all lengths. Given the rank distribution for all candidates of length k, let $\bar{R}^k$ be the average rank, and $\hat{S}^k$ be the standard deviation. Then given a sequence of length k, with rank r, the normalized rank will be $\frac{r-\bar{R}^k}{\hat{S}^k}$. This translates the rank r into units of standard deviation, where positive values are above average. Using the normalized rank, one can compare pattern candidates of different lengths, since all normalized ranks are in units of standard deviation.

4.2. Generalizing from Similar Patterns

A second limitation we have found in existing methods is inability to generalize patterns, in the sense that minor variations on the same pattern would be treated as completely different patterns by the learning methods. For instance, if a pattern $ABCD$ was ranked high, the algorithm was likely to also rank high the *shadow sub-patterns* ABC, BC, BCD, etc.

We focus on a clustering approach, in which we group together pattern variations. We use a clustering algorithm to group candidates that are within a user-specified threshold of *edit distance* from each other. The procedure goes through the list of candidates top-down. The first candidate is selected as the representative of the first cluster. Each of the following candidates is compared against the representatives of each of the existing groups. If the candidate is within a user-provided edit-distance from a representative of a cluster, it is inserted into the representative's group. Otherwise, a new group is created, and the candidate is chosen as the group's representative. The result set is composed of all the group representatives.

Generally, the edit-distance between two sequences is the minimal number of editing operations (insertion, deletion or replacement of a single event) that should be applied on one sequence in order to turn it into the other. For example, the editing distance between ABC and ACC is 1, as is the editing distance between AC and ABC. A well known method for calculating the edit distance between sequences is *global alignment* [10]. However, our task requires some modifications to the general method. For example, the sequence pairs $\{ABCDE, BCDEF\}$ and $\{ABCD, AEFD\}$ have an edit-distance of 2, though the former pair has a large overlapping subsequence ($BCDE$), and the latter pair has much smaller (fragmented) overlap $A??D$.

We use a combination of a modified (weighted) distance calculation, and heuristics which come to bear after the distance is computed. Our alignment method classifies each event (belonging to one sequence and/or the other) as one of three types: appearing before an overlap between the patterns, appearing within the overlap, or appearing after the overlap. It then assigns a *weighted* edit-distance for the selected alignment, where the edit operations have weights that differ by the class of the events they operate on. Edit operations within the overlap are given a high weight (called *mismatch weight*). Edit operations on events appearing before or after the overlap are given a low weight (*edge weight*). In our experiments we have used an infinite mismatch weight, meaning we did not allow any mismatch within the overlapping segment. However, both weight values are clearly domain-dependent.

In order to avoid false alignments where the overlapping segment is not a significant part of the overall unification, we set a minimal threshold upon the length of the overlapping segment. This threshold is set both as an absolute value and as a portion of the overall unification's length.

5. Experiments

To evaluate the presented techniques, we conducted extensive experiments on synthetic and real data. We begin with the synthetic data. We repeated our experiments from Section 3, but this time with the modified techniques. The results are shown in Figures 3 and 4. The figures plot the precision achieved at different pattern rates, par-

alleling Figures 1 and 2. They contrast standard, normalized, and normalized-clustered versions of DD, Support, and Support-Confidence. Note that by support/confidence we refer to the optimal version that dynamically adjusts the minimal thresholds in an unrealistic manner. Figure 4 zooms on pattern rates lower than 5%.

The results show that length normalization significantly improves *all* recognition algorithms tested. For instance, normalization of support/confidence has increased hit rate from 20% up to 50% for a pattern rate of 2%.

Clustering the normalized results improved the results further, by notable margins. For instance, the clustered normalized version of support/confidence has achieved precision of above 70% for a pattern rate of 5%, comparing to less than 60% achieved by the normalized version, and 33% provided by the standard technique. Note that after length-based standardization and clustering, DD may no longer be superior over the support/confidence approach.

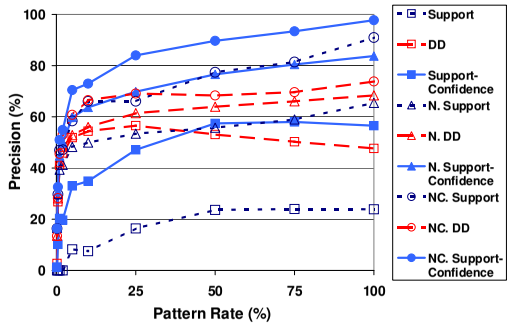

Figure 3: Modified: All Pattern Rates.

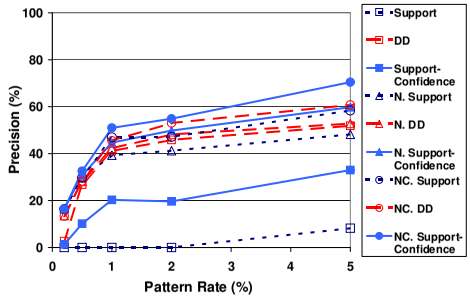

Figure 4: Modified: Low Pattern Rates.

We now turn to examine the effects of intra-pattern noise on the quality of the patterns generated by the different techniques. Figure 5 shows (y-axis) the precision resulting from using the Normalized and Normalized Clustered versions of Support, Support-Confidence, and DD. We used patterns of length 3–5 with the pattern-rate fixed at 5%, while intra-pattern noise rate varied from 0% to 60% (on the x-axis). The figure shows a distinct advantage to using clustering with all normalized methods—indeed there's a significant jump between the group of normalized methods to normalized-clustered methods. In addition, the figure shows that this advantage is maintained even when the amount of intra-pattern noise is increased, although at 25% intra-pattern noise the two distinct groups begin to cross over in terms of relative performance.

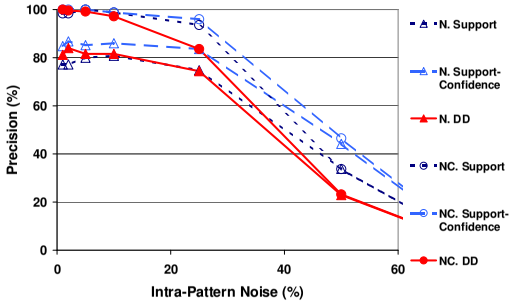

Figure 5: Handling Intra-Pattern Noise.

We also evaluated our techniques with real-world data. We used the text of George Orwell's *1984* to test our modified techniques in the following manner. We changed the original text by introducing noise within the words and between them. For instance, the first sentence in the book - "It was a bright cold day in April" was replaced by "**It**oH7l4H XywOct8M (...9 more noisy words) 6j**Owas**2x imfG8e1x (...2 more noisy words) nB**a**or1oL iWtHhTEq **bright**cT x**cold**Vuv vf**day**1Ap BsQG9pyK 8Nxf**in**XR 8TGmxcXO E1IenU2Q **Apri**u**l**xL". We inserted only fixed 8-character sequences, such that each actual word that is shorter than 8 characters was padded with noise from one or more sides. We set pattern rate to 40% by inserting 6 noisy sequences for each 4 actual words. Intra pattern noise was set at 10%. We then counted how many of the top 100 candidates returned by each technique are actual words. The results, reflecting the average precision over the first 8 chapters of the book, are shown in Figure 6.

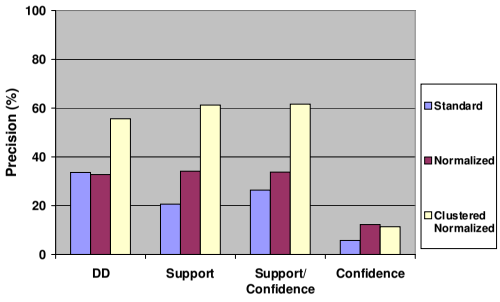

Figure 6: Precision: Orwell's 1984.

The results show that the clustered normalized versions have significantly outperformed the standard versions, increasing precision by up to 42% for the support algorithm. The normalized versions have typically outperformed the standard versions, except for the case of DD, where the nor-

malized results contained various sequences that reflected the same words (see Section 4.2), and were then significantly improved by our clustering approach. Note also that among the standard techniques, DD has once again outperformed the other methods.

We conducted additional experiments on UNIX command line sequences. In this case, we could not know in advance what true patterns were included in the data, thus quantitative accuracy comparison was not possible. However, we hoped to contrast the pattern candidates generated by the different methods. We have analyzed 9 data sets of UNIX command line history [4]. The data reflects command histories of 8 different users at Purdue over the course of up to 2 years. These tests have also suggested that DD was superior to other techniques. While the results of support and confidence methods consisted mainly of different variations of ls, cd and vi, DD was the only algorithm to discover obviously-sequential patterns (that were not frequent) such as "*g++ -g <file> a.out*", "*| more*", "*ls -al*", "*ps -ef*", "*xlock -mode*", "*pgp -kvw"*, and more.

The clustered normalized versions of both DD and support/confidence were able to detect complex user patterns, such as:

1. *ps -aux | grep <process>; kill -9*—a user looking for a certain process id to kill.
2. *tar <3 args>; cd; uuencode <2 args> > <file>; mailx*—a user packaging a directory tree, encoding it to a file, and sending it by mail.
3. *compress <arg>; quota; compress <arg>; quota*—a user trying to overcome quota problems by compressing files.
4. *latex <arg>; dvips <arg>; ghostview*—a latex $write \rightarrow compile \rightarrow view$ cycle.
5. *vi <arg>; gcc <arg>; a.out; vi <arg>; gcc*—an $edit \rightarrow compile \rightarrow run$ cycle.

6. Related Work

Section 2 has already introduced some of the key sequence recognition algorithms. Following its introduction in [1], much effort has been devoted to improving the performance of support-confidence for data mining, both in terms of quality of results as well as the computational complexity of the underlying algorithms (e.g., [12]). The quality-enhancing methods we present are not necessarily appropriate for very large databases, and thus their use in data mining may be limited at this stage. Some methods for improving quality in data mining rely on a domain expert to provide target templates (e.g., [8]), which would be inappropriate given our emphasis on "hands-free" learning.

Clustering is used by Bauer [2] to learn users' web-usage plans. In our technique, we first extract common ordered sequential patterns, and then cluster them. Bauer's technique first clusters observed segments based on similarity, with no regard for order. The clusters are used as classes for a supervised learning algorithm which attempts to extract the actions common to all streams within a cluster.

The augmented suffix-trie representation was used in [7] for learning the sequential coordinated behavior of RoboCup soccer teams from observations, by applying both DD and support. However, no normalization or clustering was used.

7. Conclusions and Future Work

This paper addresses learning of the sequential behavior of observed agents. First, it empirically compares unsupervised sequence learning algorithms and finds that dependency-detection methods outperform frequency-based techniques. The results also show several common deficiencies in all tested algorithms: All are susceptible to a bias in preferring pattern candidates based on length and all fail to generalize similar results patterns.

We present a normalization method to effectively neutralize the length bias, leading to precision improvements by up to 42% in our experiments. We then present a clustering approach, based on a weighted edit-distance measure, to group together all patterns that are closely related. The use of clustering in addition to normalization had further improved precision by up to 25%. Finally, the improved methods were shown to be robust to noise, and were successfully applied to real-world data sets: sequences from Orwell's 1984, and UNIX command-line data.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[2] M. Bauer. From interaction data to plan libraries: A clustering approach. In *IJCAI-99*, volume 2, pages 962–967, Stockholm, Sweden, August 1999. Morgan-Kaufman Publishers, Inc.

[3] P. Cohen and N. Adams. An algorithm for segmenting categorical time series into meaningful episodes. *Lecture Notes in Computer Science*, 2189, 2001.

[4] S. Hettich and S. D. Bay. The uci kdd archive. http://kdd.ics.uci.edu/, 1999.

[5] A. E. Howe and P. R. Cohen. Understanding planner behavior. *AIJ*, 76(1–2):125–166, 1995.

[6] A. E. Howe and G. L. Somlo. Modeling intelligent system execution as state-transition diagrams to support debugging. In *Proceedings of the Second International Workshop on Automated Debugging*, May 1997.

[7] G. A. Kaminka, M. Fidanboylu, A. Chang, and M. Veloso. Learning the sequential behavior of teams from observations. In *Proceedings of the 2002 RoboCup Symposium*, 2002.

[8] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In N. R. Adam, B. K. Bhargava,

and Y. Yesha, editors, *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 401–407. ACM Press, 1994.

[9] D. E. Knuth. *Sorting and Searching*, volume 3 of *The art of computer programming*. Addison-Wesley, 1973.

[10] P. Sellers. The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, (1):1:359–373, 1980.

[11] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: Generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery*, 2(1):39–68, 1998.

[12] M. Zaki. Fast mining of sequential patterns in very large databases. Technical Report 668, University of Rochester Compute Science Department, 1997.