Removing Statistical Biases in Unsupervised Sequence Learning

Yoav Horman and Gal A. Kaminka

The MAVERICK Group
Department of Computer Science
Bar-Ilan University, Israel
{hormany,galk}@cs.biu.ac.il

Abstract. Unsupervised sequence learning is important to many applications. A learner is presented with unlabeled sequential data, and must discover sequential patterns that characterize the data. Popular approaches to such learning include statistical analysis and frequency based methods. We empirically compare these approaches and find that both approaches suffer from biases toward shorter sequences, and from inability to group together multiple instances of the same pattern. We provide methods to address these deficiencies, and evaluate them extensively on several synthetic and real-world data sets. The results show significant improvements in all learning methods used.

1 Introduction

Unsupervised sequence learning is an important task in which a learner is presented with unlabeled sequential training data, and must discover sequential patterns that characterize the data. Applications include user modeling [1], anomaly detection [2], data-mining [3] and game analysis [4].

Two popular approaches to this task are frequency-based (*support*) methods (e.g., [3]), and statistical dependence methods (e.g., [5]—see Section 2 for background). We empirically compare these methods on several synthetic and real-world data sets, such as human-computer command-line interactions. The results show that statistical dependence methods typically fare significantly better than frequency-based ones in high-noise settings, but frequency-based methods do better in low-noise settings.

However, more importantly, the comparison uncovers several common deficiencies in the methods we tested. In particular, we show that they are (i) biased in preferring sequences based on their length; and (ii) are unable to differentiate between similar sequences that reflect the same general pattern.

We address these deficiencies. First, we show a length normalization method that leads to significant improvements in *all* sequence learning methods tested (up to 42% improvement in accuracy). We then show how to use clustering to group together similar sequences. We show that previously distinguished sub-patterns are now correctly identified as instances of the same general pattern, leading to additional significant accuracy improvements. The experiments show that the techniques are generic, in that they significantly improve all of the methods initially tested.

2 Background and Related Work

In unsupervised learning of sequences, the learner is given example streams, each a sequence $\alpha_1, \alpha_2, \ldots, \alpha_m$ of some atomic *events* (e.g., observed actions). The learner must extract sequential *segments* (also called *patterns*), consecutive subsequences with no intervening events, which characterize the example streams. Of course, not every segment is characteristic of the streams, as some of the segments reflect no more than a random co-occurrence of events. Moreover, each observation stream can contain multiple segments of varying length, possibly interrupted in some fashion. Thus it is important to only extract segments that signify invariants, made up of events that are predictive of each other. We provide the learner with as little assistance as possible.

The literature reports on several unsupervised sequence learning techniques. One major approach learns segments whose frequency (*support*) within the training data is sufficiently high [3]. To filter frequent segments that are due to chance—segments that emerge from the likely frequent co-occurrence of a frequent suffix and a frequent prefix—support-based techniques are usually combined with *confidence*, which measures the likelihood of a segment suffix given its prefix. In such combinations, the extracted segments are those that are more frequent than a user-specified *minimal support* threshold, and more predictive than a user-specified *minimal confidence*.

Another principal approach is statistical dependency detection (*DD*) [5]. DD methods test the statistical dependence of a sequence suffix on its prefix, taking into account the frequency of other prefixes and suffixes. To calculate the rank of a given segment S of size k, a 2×2 contingency table is built for its $(k-1)$-prefix p_r and suffix α_k (Table 2). In the top row, n_1 reflects the count of the segment S. n_2 is the number of times we saw a different event following the same prefix, i.e., $\sum_{i\neq k} count(p_r\alpha_i)$. In the second row, n_3 is the number of segments of length k in which α_k followed a prefix different than p_r ($\sum_{S\neq p_r,|S|=|p_r|} count(S\alpha_k)$). n_4 is the number of segments of length k in which a different prefix was followed by a different suffix ($\sum_{S\neq p_r,|S|=|p_r|} \sum_{i\neq k} count(S\alpha_i)$). The table margins are the sums of their respective rows or columns. A chi-square or G test [6] is then run on the contingency table to calculate how significant is the dependency of α_k on p_r. This is done by comparing the observed frequencies to the expected frequencies under the assumption of independence. DD methods have been utilized in several data analysis applications, including analysis of execution traces [5], time-series analysis [7], and RoboCup soccer coaching [4].

Table 1. A statistical contingency table for segment S, composed of a prefix $p_r = \alpha_1, \alpha_2, \ldots, \alpha_{k-1}$ and a suffix α_k. In all cases, $|S| = |p_r|$.

	α_k	$\neg\alpha_k$	
p_r	n_1	n_2	$\sum_i count(p_r\alpha_i)$
$\neg p_r$	n_3	n_4	$\sum_{S\neq p_r,i} count(S\alpha_i)$
	$\sum_S count(S\alpha_k)$	$\sum_{S,i\neq k} count(S\alpha_i)$	

We focus in this paper on support/confidence and DD. However, other technique exist (see [8] for a survey), including statistical methods such as interest ([9]) and conviction ([10]) that are often combined with support. We have successfully applied our methods to these, but do not discuss them here for lack of space. In addition, Likewise, we ignore here methods requiring human expert guidance or other pre-processing.

Also related to our work are [1] and [2], which use clustering techniques to learn the sequential behavior of users. A common theme is that clustering is done based on similarity between sequential pattern instances in the training data. Bauer then uses the resulting clusters as classes for a supervised learning algorithm. Lane and Brodley use the clusters to detect anomalous user behavior. Neither investigates possible statistical biases as we do.

3 A Comparison of Unsupervised Techniques

We conducted extensive experiments using synthetic data, comparing support, confidence, support/confidence and dependency-detection using a G-test. In each run, the techniques above were to discover five different re-occurring *true segments*, uniformly distributed within a file of 5000 streams. We refer to the percentage of the streams that contain true segments as *pattern rate*; thus low pattern rates indicate high levels of noise. The example streams might include additional random events before, after, or within a segment. We controlled *intra-pattern noise rate*: the probability of having noise inserted within a pattern.

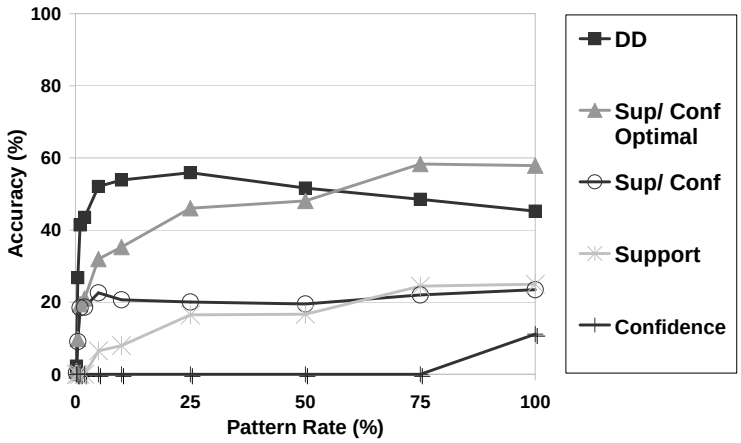

Fig. 1. Accuracy of unsupervised sequence learning methods.

In each experiment, each technique reported its best 10 segment candidates, and those were compared to the five true segments. The results were measured as the percentage of true segments that were correctly detected (the recall of the technique, hereinafter denoted *accuracy*). 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”). While the support/confidence method is meant to return all segments satisfying the thresholds, with no ordering, we approximated ranking of resulting segments by their support (after thresholding).

We varied several key parameters in order to verify the consistency of the results. For three different values of alphabet size, denoted T (5, 10 and 26) and three ranges of true-pattern sizes (2–3, 3–5 and 4–7) we have generated data sets of sequences with incrementing values of pattern rate. 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 different sets of 5 true patterns.

The results are depicted in Figure 1. 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. Each point in the figure reflects the average of *450* different tests. The “Optimal Support/Confidence” technique is denoted "Sup/Conf Optimal", where the standard method, using a fixed minimal confidence value, is denoted "Sup/Conf". The dependency-detection is denoted “DD”.

The figure shows that dependency-detection (*DD*) outperforms all other methods for low and medium values of pattern rate. However, the results cross over and support/confidence optimal outperforms DD at high pattern rates. The standard support/confidence, as well as the simple support technique, provide relatively poor results. Finally, confidence essentially fails for most pattern rate values.

Figure 2 shows the results for the same experiment, focusing on pattern rates up to 5%. As can be clearly seen, DD quickly achieves relatively high accuracy, at least twice as accurate as the next best technique, support/confidence optimal. A paired one-tailed t-test comparing DD and support/confidence optimal for pattern rates of up to 5% shows that the difference is significant at the 0.05 significance level ($p < 1 \times 10^{-10}$).

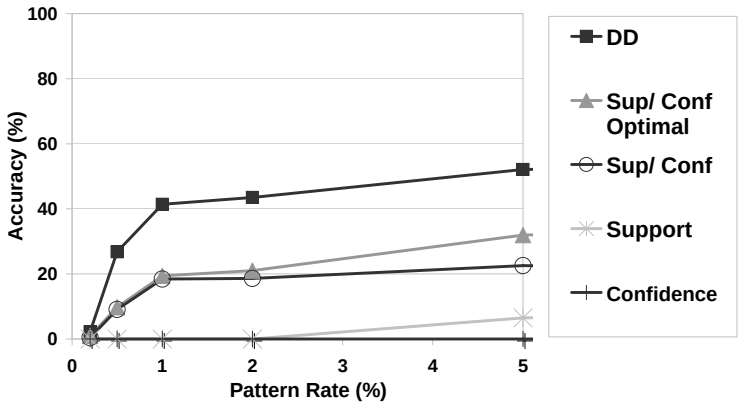

Fig. 2. Accuracy at low pattern rates (high noise).

For lack of space, we only briefly discuss the effect of the alphabet size T on the accuracy of the different algorithms. All methods achieve better results with greater alphabet sizes. However, when the alphabet is small ($T = 5$), the results of DD are up to 25 times more accurate than other methods, in high noise settings.

4 Statistical Biases

We analyzed the results of the different techniques and found that all methods suffer from common limitations: (i) Bias with respect to the length of the segments (Section 4.1); and (ii) inability to group together multiple instances of the same pattern (4.2).

4.1 Removing the Length Bias

The first common limitation of the approaches described above is their bias with respect to the length of the segments. Figure 3 shows the average length of the segments returned by the learning algorithms, in a subset of the tests shown in Figure 1, for two different values of pattern rate (0.5% and 75%), where the length of the true patterns was set to 3–5 (average ≈ 4) and alphabet size was fixed at 10. The figure shows that the support algorithm prefers short segments. The optimal support/confidence algorithm behaves similarly, though it improves when pattern rate increases (75%). DD is slightly better, but also prefers shorter sequences at low noise (high pattern rate) settings. In contrast to all of these, Confidence prefers longer sequences.

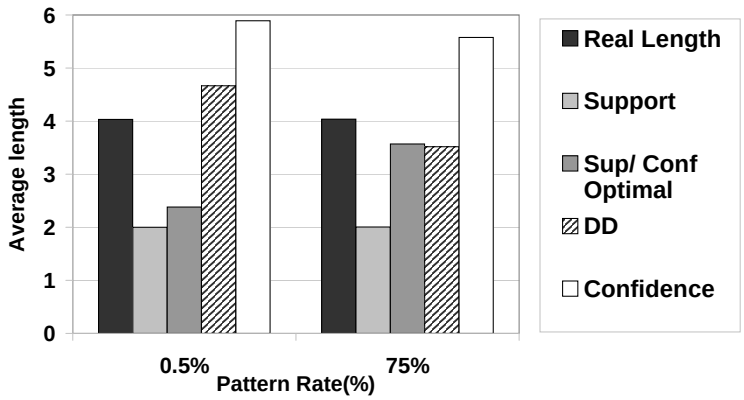

Fig. 3. Average segment length for two pattern rate values.

Different methods have different reasons for these biases. Support-based methods have a bias towards shorter patterns, because there are more of them: Given a target pattern *ABCD*, the pattern *AB* will have all the support of *ABCD* with additional support from (random) appearances of *ABE,ABC,ABG,*.... Confidence has a bias towards longer

sequences, because their suffix can be easily predicted based on their prefix simply because both are very rare. Finally, DD methods prefer shorter segments at higher pattern rate settings. We found that this is due to DD favoring subsequences of true patterns to the patterns themselves. When pattern rate is high, significant patterns also have significant sub-patterns. Even more: The sub-patterns may have higher significance score because they are based on counts of shorter sequences—which are more frequent as we have seen. This explains the degradation in DD accuracy at higher pattern rates.

In order to overcome the length bias obstacle, we normalize candidate pattern ranks based on their length. 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 of ranks. 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. This method was used in [7] for unsupervised segmentation of observation streams based on statistical dependence tests.

4.2 Generalizing from Similar Patterns

A second limitation we have found in existing methods is inability to generalize patterns, in the sense that sub-segments of frequent or significant patterns are often themselves frequent (or significant). Thus both segment and its subsegment receive high normalized ranks, yet are treated as completely different patterns by the learning methods. For instance, if a pattern $ABCD$ is ranked high, the algorithm is likely to also rank high the *shadow sub-patterns* ABC, BC, etc. Normalizing for length helps in establishing longer patterns as preferable to their shadows, but the shadows might still rank sufficiently high to take the place of other true patterns in the final pattern list.

We focus on a clustering approach, in which we group together pattern variations. We cluster 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 its representative. The result set is composed of all 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* [11]. 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 techniques we presented, we conducted extensive experiments on synthetic (Section 5.1) and real data (5.2).

5.1 Synthetic Data Experiments

We repeated our experiments from Section 3, this time with the modified techniques. Figure 4 and Table 2 show the accuracy achieved at different pattern rates, paralleling Figures 1 and 2, respectively. Figure 4 shows all results, while Table 2 focuses on low pattern rates (high noise). Every point (table entry) is the average of *450* different tests, contrasting standard, normalized (marked *N*) and normalized-clustered (*NC*) versions of DD, Support, and Optimal Support/Confidence (marked simply as Sup/Conf).

The results show that length normalization improves *all* tested algorithms. For instance, the support technique has completely failed to detect true segments for a pattern rate of 1%, while its normalized version has achieved accuracy of 39% at this rate. Clustering the normalized results improved the results further, by notable margins.

The improvements derived from normalizing and clustering the results both proved to be statistically significant for all learning techniques. For instance, a paired one-tailed t-test shows that the normalized version of DD is significantly better than the standard version ($p < 1 \times 10^{-10}$) and that the clustered-normalized version of DD significantly outperforms the normalized version ($p < 1 \times 10^{-10}$).

The improvements are such that after normalization and clustering, the simple support technique outperforms the standard DD method for all pattern rate values, including 0.2%-0.5%. This is where standard DD performs significantly better than the other standard techniques. Indeed, after length-based standardization and clustering, DD may no longer be superior over the support/confidence approach.

Figure 5 shows the results from one specific setting, where both normalizing and normalizing-clustering proved particularly effective. Each point in the figure represents

Table 2. Accuracy at low pattern rates.

Pattern Rate (%)	0.2	0.5	1	2	5	10
DD	2.7	26.8	41.1	45.9	51.9	54.4
N. DD	**13.6**	28.1	42.3	48.2	52.8	56.0
NC. DD	13.4	**28.3**	**45.6**	**53.0**	**60.8**	**66.5**
Sup/ Conf	1.3	10.2	20.3	19.6	33.0	34.9
N. Sup/ Conf	**16.6**	30.8	44.7	49.8	59.8	63.7
NC. Sup/ Conf	16.4	**32.6**	**51.0**	**54.9**	**70.4**	**73.0**
Sup	0.0	0.0	0.0	0.0	8.2	7.5
N. Sup	**16.6**	**29.9**	39.3	41.2	48.2	50
NC. Sup	16.4	29.7	**47.0**	**47.1**	**58.4**	**66.0**
Conf	0.0	0.0	0.0	0.0	0.0	0.0
N. Conf	0.0	1.0	13.4	21.3	31.0	37.1
NC. Conf	0.0	**1.2**	**14.4**	**22**	**34**	**42.0**

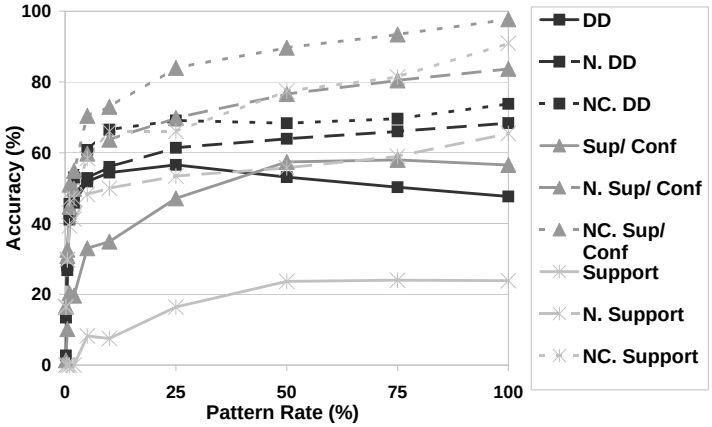

Fig. 4. Modified: All Pattern Rates, average results.

the average of 50 different tests, with an alphabet size 26, and true patterns composed of 3–5 events. In the figure, the normalized version of the support technique has achieved accuracy of 78% for a pattern rate of 1%, comparing to 0% accuracy of the standard version. The normalized clustered versions of all algorithms have achieved more than 95% accuracy for a pattern rate as low as 1%, where the accuracy of the standard techniques was 0% for support, 66% for support/confidence and 82% for DD.

We have also evaluated the effects of intra-pattern noise on the quality of results. In general, for all alphabet and pattern sizes, we have found that the Normalized Clustered versions offer consistent improvements to accuracy of Normalized methods, in the presence of up to 25% intra-pattern noise.

We evaluated our techniques with an additional dataset. We used the text of George Orwell's *1984* to test our modified techniques on data that was both more realistic, yet

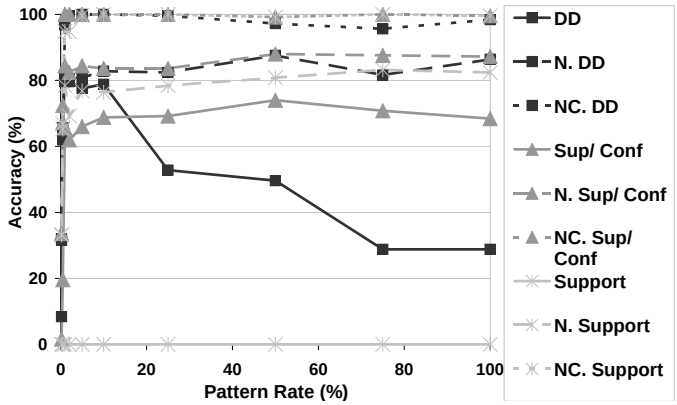

Fig. 5. Modified: $T = 26$**, Patterns of size 3–5**

still allowed for controlled experiments. In one experiment, 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) 6jO**was**2x imfG8e1x (...2 more noisy words) nB**a**or1oL iWtHhTEq **bright**cT x**cold**Vuv vf**day**1Ap BsQG9pyK 8Nxf**in**XR 8TGmx-cXO 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, and words longer than 8 characters were cut. We set pattern rate to 40% by inserting 6 noisy streams, for each 4 containing 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 appearing in the book. We hoped to find as many actual words in the results set as possible. The results, reflecting the average accuracy over the first 8 chapters of the book, are shown in Figure 6. Similar improvement results were achieved for other settings of pattern rate and intra pattern noise.

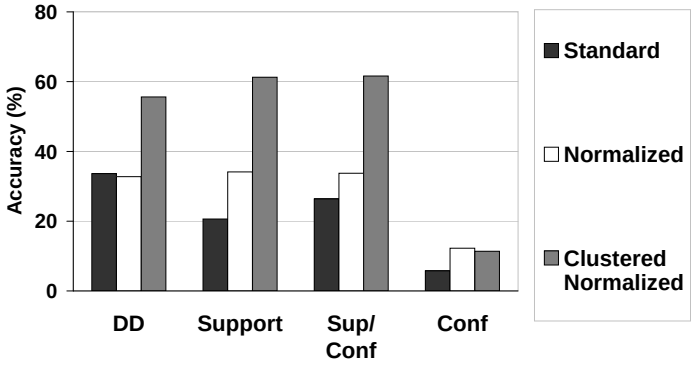

Fig. 6. Accuracy improvements: Orwell's 1984.

The results show that for each of the presented techniques the clustered normalized versions have significantly outperformed the standard versions, increasing accuracy by up to 41% for the support algorithm. The normalized versions have typically outperformed the standard versions, except for the case of DD, where the normalized 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.

5.2 Real World Experiments

We conducted real-world experiments on UNIX command line sequences. We utilized 9 data sets of UNIX command line histories, collected for 8 different users at Purdue university over the course of 2 years [12]. In this case, we do not know in advance what true patterns were included in the data, thus quantitative evaluation of accuracy is not possible. However, we hoped to qualitatively contrast the pattern candidates generated by the different methods.

The results suggest again that among non-normalized techniques, DD is superior. 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 necessarily frequent) such as "*g++ -g <file>;a.out*", "*| more*", "*ls -al*", "*ps -ef*", "*xlock -mode*", "*pgp -kvw*", etc.

The clustered-normalized versions of both DD and support/confidence detected more complex user patterns, which were not detected by the standard techniques. The results clearly show the ability of the improved techniques to discover valuable sequential patterns, which characterize interesting user behavior, and are overlooked by the standard methods. Among these sequential patterns are:

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 Conclusions and Future Work

This paper tackles the problem of unsupervised sequence learning. The challenge is addressed by improving sequence learning algorithms, which extract meaningful patterns of sequential behavior from example streams. We empirically compared these algorithms, to determine their relative strengths. Based on the comparison, we noted 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 patterns, often taking a high-ranked pattern candidate as distinct from its shorter sub-patterns.

We use a normalization method to effectively neutralize the length bias in all learning methods tested, by normalizing the frequency/significance rankings produced by

the learning methods. Use of this method had improved accuracy by up to 42% in testing on synthetic data. We then use a clustering approach, based on a modified weighted edit-distance measure, to group together all patterns that are closely related. The use of clustering in addition to normalization had further improved accuracy by up to 22% in some cases. We also show that the techniques are robust to noise in and out of the patterns. Finally, the improved methods were run on two additional sets of data: sequences from Orwell's 1984, and UNIX real-world command-line data. The methods successfully detected many interesting patterns in both.

A weakness with the methods that we presented is their use with very large databases. For instance, normalization requires repeatedly counting all the patterns in the database, and would therefore be inefficient for large data-mining applications. However, the techniques we presented are well suited for typical agent-observation data (such as RoboCup soccer logs or UNIX command-line data). We plan to consider large data-mining applications in our future work.

References

1. Bauer, M.: From interaction data to plan libraries: A clustering approach. In: IJCAI-99. Volume 2., Stockholm, Sweden, Morgan-Kaufman Publishers, Inc (1999) 962–967
2. Lane, T., Brodley, C.E.: Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security **2** (1999) 295–331
3. Agrawal, R., Srikant, R.: Mining sequential patterns. In Yu, P.S., Chen, A.S.P., eds.: Eleventh International Conference on Data Engineering, Taipei, Taiwan, IEEE Computer Society Press (1995) 3–14
4. Kaminka, G.A., Fidanboylu, M., Chang, A., Veloso, M.: Learning the sequential behavior of teams from observations. In: Proceedings of the 2002 RoboCup Symposium. (2002)
5. Howe, A.E., Cohen, P.R.: Understanding planner behavior. AIJ **76** (1995) 125–166
6. Sokal, R.R., Rohlf, F.J.: Biometry: The Principles and Practice of Statistics in Biological Research. W.H. Freeman and Co.,, New York (1981)
7. Cohen, P., Adams, N.: An algorithm for segmenting categorical time series into meaningful episodes. Lecture Notes in Computer Science **2189** (2001)
8. Tan, P.N., Kumar, V., Srivastava, J.: Selecting the right interestingness measure for association patterns. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press (2002) 32–41
9. Silverstein, C., Brin, S., Motwani, R.: Beyond market baskets: Generalizing association rules to dependence rules. Data Mining and Knowledge Discovery **2** (1998) 39–68
10. Brin, S., Motwani, R., Ullman, J.D., Tsur, S.: Dynamic itemset counting and implication rules for market basket data. In Peckham, J., ed.: SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, ACM Press (1997) 255–264
11. Sellers, P.: The theory and computation of evolutionary distances: pattern recognition. Journal of Algorithms (1980) 1:359–373
12. Hettich, S., Bay, S.D.: The uci kdd archive. http://kdd.ics.uci.edu/ (1999)