Prediction Algorithms for User Actions

Melanie Hartmann and Daniel Schreiber
Telecooperation Group, Darmstadt University of Technology
D-64289 Darmstadt, Germany
{melanie,schreiber}@tk.informatik.tu-darmstadt.de

Abstract

Proactive User Interfaces (PUIs) aim at facilitating the interaction with a user interface, e.g., by highlighting fields or adapting the interface. For that purpose, they need to be able to predict the next user action from the interaction history. In this paper, we give an overview of sequence prediction algorithms (SPAs) that are applied in this domain, and build upon them to develop two new algorithms that base on combining different order Markov models. We identify the special requirements that PUIs pose on these algorithms, and evaluate the performance of the SPAs in this regard. For that purpose, we use three datasets with real usage-data and synthesize further data with specific characteristics. Our relatively simple yet efficient algorithm FxL performs extremely well in the domain of SPAs which make it a prime candidate for integration in a PUI. To facilitate further research in this field, we provide a Perl library that contains all presented algorithms and tools for the evaluation.

1 Introduction

Nowadays applications gain more and more functionality, mostly leading to a decreased usability. An intelligent interface can counter this effect by explaining the user how to work with an application, by performing repetitive actions on his behalf, by adapting the interface to his needs, and by suggesting content for input fields. Proactive user interfaces (PUI) aim at combining all these features in an augmentation of traditional user interfaces. We state that the main features of a PUI are:

- **Support Mechanisms**: provide online help that adapts to the user and his current working context
- **Interface Adaptation**: adapt the provided options and content to the user's needs and preferences
- **Content Prediction**: suggest data to be entered that is inferred from previous interactions and context information
- **Task Automation**: recognize usage patterns to allow automation of repetitive tasks.

Thereby, the interface adaptation can range from highlighting the functions the user really needs in his current context to minimizing the interface to these functions. There has been a debate for years whether automatic adaptation of the user interface helps or confuses the user [Findlater and McGrenere, 2004; Sears and Shneiderman, 1994; Mitchell and Shneiderman, 1989]. Adaptation that is not explicitly asked for by the user is often seen as a flaw that introduces unpredictable behavior and that therefore should be avoided. However, [Gajos *et al.*, 2006] shows that carefully planned adaptation can indeed improve usability and avoid confusion. We assume that interface adaptation in form of highlighting interface elements does not confuse the user and can reduce his cognitive load by helping him to focus on the important parts of the interface. Furthermore, there are also scenarios, in which a complete reordering of the interface is needed. For example, mobile users often have only a small display to their disposal, which makes it essential that the shown functions are limited to those the user needs most. For these tasks, we need an algorithm that can predict the next user actions, a so called-sequence prediction algorithm (SPA). However, knowing the next user action is not only of importance for the interface adaptation, but also for supporting the user by telling him what to do next, for suggesting content for the next input field and for automating tasks.

In this paper, we examine statistical methods for sequence prediction that can be applied without any prior task knowledge. We explore the limits of such an approach to define the baseline for further enhancements. Moreover, we compare the best performing algorithms with respect to the required memory and processing time, as those factors are of great importance when used on small mobile devices. In future research, we aim at integrating knowledge about the task and the context in the SPAs. This knowledge can be provided by the application developer, learned by the PUI, and/or modified by the user. This should then also enable us to predict not only the next action, but also content for input fields what can hardly be achieved by pure statistical approaches.

There already are some papers on SPAs, but they mostly compare just one algorithm to a baseline and moreover lack a formal description of the presented algorithms. Further, they do not describe a general method for the evaluation that is applicable to different application domains with varying requirements on the SPA. In this paper, we try to overcome these limitations first by giving a broad overview of existing algorithms with their formal descriptions and second by showing how SPAs can be systematically evaluated. To facilitate further research in this field, we provide a Perl library [Hartmann and Schreiber, 2007] that contains all presented algorithms and some tools for the evaluation.

The structure of this paper is as follows. In Section 2, we provide a short definition of sequence prediction along with some specific requirements for the problem of user action prediction in PUIs. We give an overview of existing algorithms in Section 3 and present two enhanced algorithms

FxL and Adaptive FxL in Section 4. Section 5 describes a systematic approach to compare the performance of these algorithms. The findings of our experiments are reported in Section 6. The paper concludes with an outline of some issues for further research.

2 Sequence Prediction

Algorithms for sequence prediction were mainly developed in the area of data compression. Intelligent user interfaces also make use of such predictions for facilitating interaction (e.g., in an intelligent home environment [Gopalratnam and Cook, 2007]) or for assisting the user by giving explanations (e.g., adaptive tutoring system [Künzer *et al.*, 2004]).

The problem of predicting the next symbol (representing a user action) in an input sequence can be formally defined as follows: Let Σ be the set of possible input symbols and let $A = a_1...a_n$ with $a_j \in \Sigma$ be a sequence of input symbols of which the first i symbols, that is $a_1 \dots a_i$, have already been observed. A SPA decides at first whether it is able to make a prediction and if so returns the probability for each symbol $x \in \Sigma$ that x is the next element in the input sequence. These values define a conditional probability distribution P over Σ, where $P(x|a_1 \dots a_i)$ is the probability for the singleton subset of Σ containing x. Most algorithms consider only the last k elements of the input sequence for the computation (i.e., $P(x|a_1 \dots a_i) = P(x|a_{i+1-k} \dots a_i)$).

In principle, there are two ways of calculating the probabilities: on-demand or live. Algorithms using the former method maintain a data structure to compute the probabilities. They update the data structure after each symbol in the input sequence, whereas the live algorithms update the probability distributions itself. We do not explicitly address the fact that Σ may be unknown to the algorithm and assume that the probability of an unseen element is 0.

For applying a SPA in a PUI, it has to satisfy certain requirements. At first it has to meet the requirements defined for an Ideal Online Learning Algorithm in [Davison and Hirsh, 1998]. Further, the algorithm needs a high applicability. For example, if we want to reduce the user interface to the most relevant functions for mobile usage, we always need a prediction what might be used next. Due to the small screens of mobile devices, it is simply not possible to display all available functions. Moreover, if we look at adapting the interface by highlighting operations, the costs that arise from false highlighting are minimal as the user will be at most distracted for a short moment. Another requirement that PUIs pose is that the algorithm should be able to return good predictions even with a small amount of training data, as the PUI should already be able to assist the user after few usages. Further, we assume that we have to deal with a high amount of long repetitive sequences. This is motivated by the fact that we aim to equip mostly form based applications that are likely to be processed in a specific manner.

3 Existing Algorithms

In this section, we describe the main ideas of the four most prominent existing SPAs. Their implementations are included in the Perl library [Hartmann and Schreiber, 2007]. Most existing algorithms use Markov models to predict the next action. They often combine the results of several different order Markov models to obtain optimal performance.

IPAM

One of the first algorithms for predicting the next user action is IPAM [Davison and Hirsh, 1998]. It employs a first order Markov model, i.e., it bases its predictions only on the last seen symbol of the input sequence. IPAM maintains a list of unconditional probabilities $P_{ipam}(x),\ x \in \Sigma$ and a table of conditional probabilities $P_{ipam}(x|y),\ x, y \in \Sigma$ with the latter being directly used as probability distribution for predicting the next symbol, i.e., $P(x|a_1 \dots a_i) = P_{ipam}(x|a_i)$. This live algorithm can be described in a recursive formula, where the new probabilities $P'_{ipam}(x|y)$ are computed for all $x \in \Sigma$ from $P_{ipam}(x|y)$ using a weight α after observing symbol a_{i+1} in the sequence A according to following equation:

$$P'_{ipam}(x|a_i) = \begin{cases} \alpha\ P_{ipam}(x|a_i) + (1-\alpha) & \text{if } x = a_{i+1} \\ \alpha\ P_{ipam}(x|a_i) & \text{otherwise} \end{cases}$$

The probabilities $P_{ipam}(x|y)$ for $y \neq a_i$ are not updated and the same equation holds for the unconditional probabilities. If a symbol y was observed for the first time, its conditional probability distribution is initialized with $P_{ipam}(x|y) = P_{ipam}(x)\ \forall x \in \Sigma$ and its unconditional probability is initialized with 0 before updating the probabilities as above.

According to some empirical results, Davison and Hirsh recommend a value of 0.8 for α. Thus, the more recent actions have a greater impact on the predictions than older actions. The basic idea of IPAM is also used in other prediction algorithms. Thereby, the table of conditional probabilities is often extended so that the conditional probabilities depend not only on the last seen item, but also on the last two items etc.

ONISI

Gorniak and Poole argue that the last action does not provide enough information to predict the next action [Gorniak and Poole, 2000]. Hence they build an on-demand prediction model called ONISI that employs a k-nearest neighbors scheme. Thereby, they consider not only the actions performed by the user, but also the corresponding user interface states. They compute a probability distribution according to the k longest sequences in the interaction history that match the immediate history. Gorniak and Poole found that $k = 5$ was sufficient to gain optimal results in their sample application (a web application for learning AI concepts). However, some actions are strongly correlated to the state in which they are performed but do not belong to long sequences. To account for this fact, Gorniak and Poole weigh off the probability distributions determined by the current state and by the longest sequences. For that purpose they use a weighting factor α that they found to perform best with a value of 0.9.

$$P_{onisi}(x|(s_1, a_1) \dots (s_i, a_i)s_{i+1}) =$$

$$\alpha \frac{l(s_{i+1}, x)}{\sum_y l(s_{i+1}, y)} + (1-\alpha) \frac{fr(s_{i+1}, x)}{\sum_y fr(s_{i+1}, y)}$$

Thereby, $l(s, a)$ returns how often the state action pair (s, a) occurred after the longest k sequences that match the recent interaction history. In contrast, $fr(s, a)$ reflects how often action a occurred in the interface state s.

For our experiments, we have to limit the algorithm to a single interface state, as the real datasets used in the experiments have no interface states and no restriction for the order of actions.

Jacobs Blockeel

Jacobs and Blockeel [Jacobs and Blockeel, 2002] claim that the longest match in the history for the immediate history is not always the best choice for determining the probabilities of the next symbol and that the ideal length can not be known in advance. Their live algorithm builds upon IPAM but allows longer premises in the table of conditional probabilities. For that purpose, they add a step that is performed only after a correct prediction was made. If the algorithm made a correct prediction for a_{i+1} after observing the sequence $A = a_1 \dots a_i$, new entries for every C that is suffix of A and where $P_{jb}(x|C) > 0$ are added to the probability table. Let L be the longest suffix of the concatenation $C \circ a_{i+1}$ for which already an entry $P_{jb}(x|L) > 0$ exists. This probability distribution is taken as the best estimation of the new probabilities. Let P_{jb} be the probabilities that result after the IPAM update step. Next the probabilities for all new premises $C \circ a_{i+1}$ are computed as

$$P'_{jb}(x|C \circ a_{i+1}) = P_{jb}(x|L).$$

So the algorithm does not rely on a fixed order Markov model, but uses a mixed order approach to compute the probability distribution of the next element. Using the above equations, the sum of the probabilities over Σ is not always 1 and a normalization has to be performed.

ActiveLeZi

Another on-demand algorithm that considers several Markov models is ActiveLeZi [Gopalratnam and Cook, 2007]. It stores the frequency of input patterns in a trie according to the compression algorithm LZ78. To overcome some of the problems arising with LZ78, a variable length window of previously-seen symbols is used. The size of the window grows with the number of different subsequences seen in the input sequence. Let suf_l be the suffix of length $l + 1$ of the immediate interaction history A, that is $a_{i-l} \dots a_i$. The probabilities are recursively defined as follows:

$$P^0_{alz}(x|A) = \frac{fr(x)}{\sum_{y \in \Sigma} fr(x \circ y)}$$

$$P^l_{alz}(x|A) =$$

$$\frac{fr(suf_l \circ x)}{fr(suf_l)} + \frac{fr(suf_l) - \sum_{y \in \Sigma} fr(suf_l \circ y)}{fr(suf_l)} P^{l-1}_{alz}(x|A)$$

Thereby, $fr(x)$ returns the frequency of the input pattern x as stored in the trie. The probability distribution that is finally returned by ActiveLeZi is P^k_{alz} where k is the current size of the window.

4 FxL and Adaptive FxL

We agree with most papers that knowing only the last action is not sufficient for making good predictions. Therefore, we tested two further on-demand approaches for combining the results from different order Markov models. The presented algorithms build on the KO algorithm [Künzer *et al.*, 2004] but vary in the weighting functions used. Further, the KO algorithm does not achieve high applicability values with the proposed parameters as it takes only frequencies of subsequences into account that have a minimal support in the input sequence.

The algorithm builds upon an n-gram trie containing the frequencies of different input subsequences. The function $fr(a_1 ... a_i)$ returns how often the sequence $a_1 ... a_i$ has already been seen. To reduce the amount of data that needs to be stored, only n-grams of a length up to a specified value k are taken into account. The n-gram models are then used to assign a score to every symbol denoting the probability of a symbol to occur next in the input sequence. As the scores for the symbols can sum up to a value greater than 1, they have to be normalized. Thus

$$P(x|a_1 ... a_i) = \frac{score(x)}{\sum_{y \in \Sigma} score(y)}$$

where $score(x)$ is calculated by adding the absolute frequencies of x succeeding any suffix (up to length $k - 1$) of $a_1 ... a_i$. As the longer suffixes yield more reliable results than the shorter ones, the frequencies are assigned a weight $w(j)$ depending on the length j of the suffix that is considered. Thus, the score is computed as follows:

$$score(x) = \sum_{j=1}^{k-1} w(j) fr(a_{i+1-j} ... a_i \circ x)$$

For $w(j)$ we tested two different weight measures. At first we tried a very simple approach that uses the suffix-length as weight, $w(j) = j$. We call this approach **FxL** as the score for a symbol is calculated by simply multiplying the frequency (F) of the symbols with the length (L) of the suffix they succeed.

However, such an approach does not adapt to the specific features of a dataset (e.g., that many shorter sequences occur in the dataset and thus need a greater weight assigned). Therefore, we developed a new algorithm called **Adaptive FxL** that considers the predictive quality of the different order models. We define the predictive quality q_i of the i-th order model as the percentage of how often it was able to make a correct prediction. For that purpose, we store for each model how often it was able to make a prediction at all (applicability ap) and how often this prediction was correct (c). The predictive quality of a model then results in $q = c/ap$. As we found that assigning greater weights to longer suffixes leads to better results, we did not use the predictive quality alone as weight, but also considered the suffix-length and an additional factor f_j: $w(j) = j\ q_j f_j$. Thereby, the factor f_j reflects the probabilities that all higher order models make a wrong prediction. Thus, the factor f_j is 1 for the highest order model that is able to make a prediction for the current sequence, and is reduced for the lower models with $f_{j-1} = (1 - q_j) f_j$.

5 Evaluating SPAs

For evaluating the performance of SPAs the following metrics are used in the literature: prediction accuracy pr_{ac}, prediction probability pr_p, and applicability ap. The prediction accuracy and probability are computed by assigning a score to every prediction made by the algorithm and averaging over the number of predictions made by the algorithm. Thus we can compute the prediction accuracy pr_{ac} and probability pr_p over the whole input sequence $a_1 ... a_n$ using following equation:

$$pr_x(a_1 ... a_n) = \frac{1}{m} \sum_{i=0}^{n-1} eval_x(a_1 ... a_i, a_{i+1})$$

where m ($\leq n$) is the number of predictions made and $eval_x$ the evaluation function that returns the value for a single prediction (it returns 0 if no prediction was made by the algorithm). The $eval_x$ function thereby differs for the two metrics:

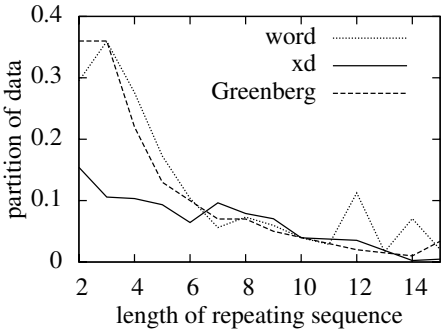

Figure 1: Distribution of repetitive sequences for three real datasets used for the evaluation

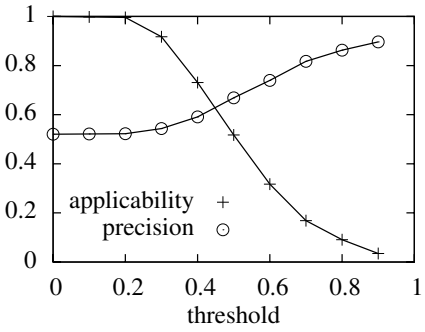

Figure 2: Applicability and precision for different confidence thresholds

The prediction accuracy $eval_{ac}$ considers only the symbols which are predicted with maximal probability. We define the set $\hat{A}$ as the set of all these symbols: $\hat{A}_{i+1} = \{x \in \Sigma | \forall y \in \Sigma . P(x|a_1 \dots a_i) \geq P(y|a_1 \dots a_i)\}$. The $eval_{ac}$ function for the prediction accuracy then returns a value reflecting whether the actual next symbol a_{i+1} is among these values $\hat{A}_{i+1}$. Thus, $eval_{ac}$ is 0 if $a_{i+1} \notin \hat{A}_{i+1}$ otherwise it is computed as $eval_{ac}(a_1 ... a_i, a_{i+1}) = \frac{1}{|\hat{A}_{i+1}|}$.

In contrast, the $eval_p$ function for the prediction probability rates with which probability the correct symbol was suggested, no matter if it has been assigned a maximal probability or not: $eval_p(a_1 ... a_i, a_{i+1}) = P(a_{i+1}|a_1 ... a_i)$.

Albrecht [Albrecht *et al.*, 1998] states that the prediction probability provides finer-grained information about the performance of an algorithm than the prediction accuracy, but that both measures produce generally consistent assessments.

Finally, the third metric applicability ap is defined as the ratio of input symbols the algorithm was able to make a prediction for: $ap = m/n$.

As the algorithms are often evaluated on datasets containing the input sequences of several users, the results have to be averaged over all users. This can be done by computing the average over the results of all users weighing every user equally, independent of the length of the corresponding sequence (macroaverage), or by averaging over all data (microaverage), emphasizing frequent users.

In order to judge which algorithm best suits the requirements of an application, we need to evaluate the algorithms depending on several parameters influencing their performance, mostly characteristics of the input sequences. The most important parameters are: (1) dataset size available for training, (2) distribution of repetitive sequences and (3) noise in the repetitive sequences.

We call a sequence repetitive if it is not part of a longer repetitive sequence and occurs at least k times in the corresponding dataset. The parameter k can be a constant or defined relatively to the dataset size. For our experiments we set $k = 5$. Figure 1 shows the sequence distribution for the real datasets Word, XD, and Greenberg used in the experiment. Noise is introduced in a repetitive sequence if the user sometimes alters the sequence of actions. However, to measure noise the repetitive sequences have to be known in advance.

To reduce the target metrics that have to be evaluated for judging the performance of an algorithm, the applicability can be turned into the fourth parameter. This also enables a better comparability of the algorithms, as they can vary heavily in their applicability values. For measuring the prediction accuracy and prediction probability for a given applicability, several (ap, pr) pairs have to be computed to infer the pr value corresponding to the specified ap value from them. The (ap, pr) pairs can be obtained by evaluating the algorithm using several thresholds for the reported predictions. This means that only predictions are taken into account whose probability is over this threshold. We assumed (and also found) that a higher threshold (up to 80 or 90%) leads to a decrease in applicability and to an increase in the prediction accuracy of the algorithm (see Figure 2). If the threshold is relatively high (over 80 or 90%) the algorithm is often only able to make very few predictions for the whole input sequence which makes the prediction value unreliable. However, this does not corrupt the results, as such low applicability values are hardly ever of interest. The required pr value is finally computed by linear interpolation of the (ap, pr) values.

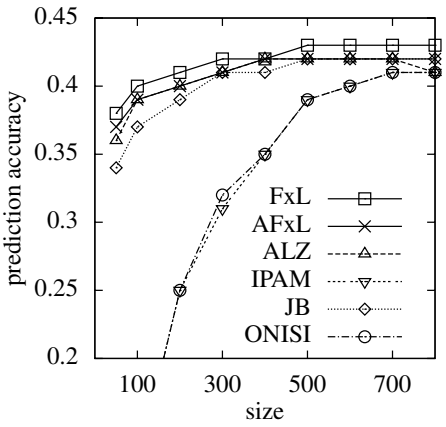

Figure 3: Performance on the Greenberg dataset regarding dataset size

In order to facilitate the evaluation, we provide a Perl library [Hartmann and Schreiber, 2007] containing the implementation of all algorithms presented in this paper. It can easily be extended with new algorithms and contains special support for facilitating the implementation of n-gram based approaches. Further, this library contains methods that compute the prediction probability or accuracy values for these algorithms.

6 Experiment

For the evaluation of the SPAs we used three real usage datasets (Greenberg, XD and Word) as well as synthesized data. The Greenberg dataset [Greenberg, 1988] is widely

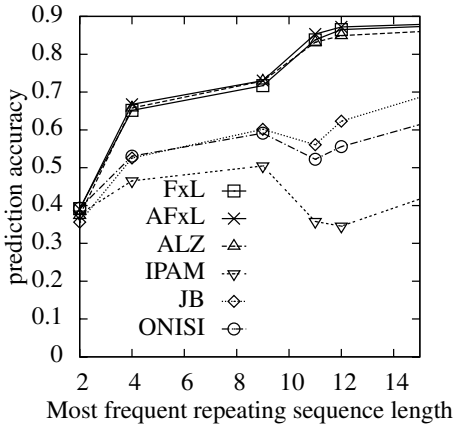

Figure 4: Performance regarding sequence length

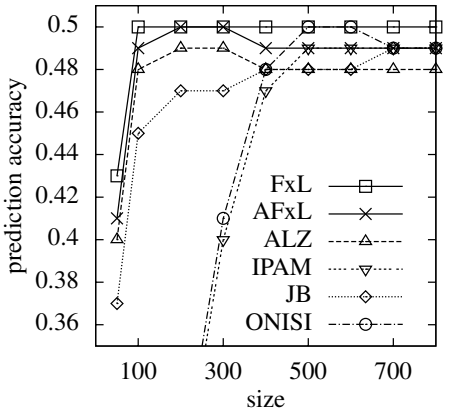

Figure 5: Performance on the Word dataset regarding dataset size

used in the literature and contains over 225 000 UNIX commands from 168 users assigned to four groups depending on their computer experience. The CrossDesktop[1] dataset (abbreviated XD) contains log data from a web application for managing files and emails. It contains about 200 000 requests from 37 users whereat the usage varies heavily between the different users. The third dataset (Word) contains logs of MS Word usage[2] described in [Linton *et al.*, 2000]. The synthesized datasets were generated to evaluate the algorithms with respect to the parameters "distribution of repetitive sequences" and "noise in repetitive sequences" that can not be varied in the real datasets. For all datasets we used the first 20% for training the algorithms, where the predictions were not included in the performance evaluation, if not explicitly stated otherwise. We chose to macroaverage the results for the different users because it is more important to optimize the results for the average than for the frequent user in our application domain . Further, we used only prediction accuracy as evaluation metric for presented results because we are more interested in the prediction with the highest probability than in its actual probability value. However, we yield consistent results if we use prediction probability or if we microaverage the results. As discussed in Section 5, we analyze the performance of the algorithms regarding the four parameters training size, distribution of repetitive sequences, noise, and applicability. Due to space limitations, we do not go into detail regarding the applicability of the algorithms, we just normalize the applicabilities as described in Section 5 to ensure the comparability of the algorithms. In the presented results, we used applicability levels of 90%.

[1]http://www.crossdesktop.com/

[2]http://www.cs.rutgers.edu/ml4um/datasets/

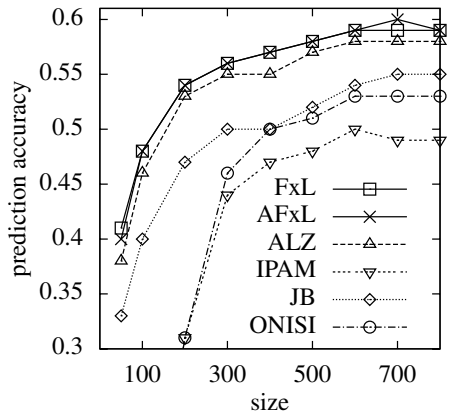

Figure 6: Performance on the XD dataset regarding dataset size

At first we calculated the prediction accuracies for the different algorithms depending on the sequence length. Thereby we included all predictions in the performance evaluation (no training data). As the results in Figure 5-3 show, FxL and Adaptive FxL (AFxL) perform best on all datasets followed by ActiveLeZi (ALZ).

In Section 5, we already pointed out that the three datasets differ in the underlying sequence distributions. To analyze the algorithms regarding this parameter, we created random datasets that varied in this parameter (using sequence lengths of 500 symbols). Figure 4 shows the average prediction accuracies over 80 randomly generated datasets. Thereby the x-coordinate represent the sequence lengths that were repeated most often in the dataset. The result is similar to the previous ones: The algorithm FxL, Adaptive FxL and ActiveLeZi perform better than the other ones.

At last, we have to consider the parameter noise for the evaluation. Thus, we also synthesized datasets that varied in their noise level. We used three operators to insert noise: An element from the original sequence could be left out, repeated up to 5 times or swapped with the following element. The versions of the dataset differed with which a noise operator was applied. The results in Figure 7 show the decay in the algorithms' performances with rising noise levels and confirm our former results.

To sum it up, we evaluated all algorithms varying the four parameters applicability, dataset size, sequence distribution and noise level using real and synthetic datasets. The overall result shows that the algorithms FxL, Adaptive FxL and ActiveLeZi perform best regarding all tested parameters.

For choosing the optimal candidate for the integration in a PUI, we also have to consider the storage and computational costs. These factors play a significant role in PUIs, as they are also used on mobile devices with memory and power restrictions. The storage costs are limited for the FxL and AFxL algorithms by the specified k and the amount of possible user actions ($|\Sigma|$), whereas the storage costs of ActiveLeZi grow with the dataset size. For our

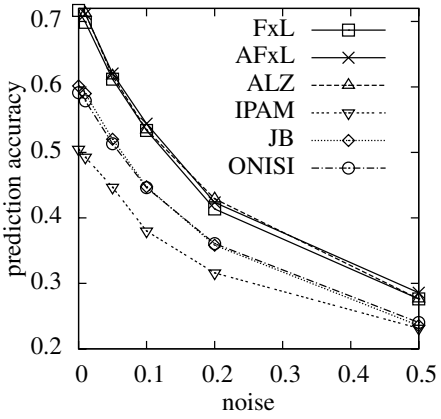

Figure 7: Performance of the algorithms regarding different noise levels

		unix	Word	XD
prediction accuracy	FxL	44.1%	57.8%	50.4%
	AFxL	43.9%	58.2%	50.1%
	ALZ	42.7%	56.0%	48.5%
stored keys	FxL:3	258	227	111
	FxL:6	2061	1599	949
	ALZ	2706	2743	1955
comput. time [s]	FxL	0.57	0.54	0.32
	AFxL	1.08	1.08	0.68
	ALZ	1.47	1.46	0.89

Figure 8: Average performance values for the three real datasets

tests we used AFxl and FxL with $k = 5$, but we found that the algorithms already reach their optimum with $k = 3$ or $k = 4$ and stay at that level for larger k. Figure 8 lists the average costs and prediction accuracies of the three prime candidates for the three real datasets. Thereby, "stored keys" reflects the keys contained in the main data structure used by the algorithm. As FxL and AFxL rely on the same data structure and vary with the specified k, we list values for the FxL algorithms with $k = 3$ and $k = 6$. The results show that the FxL algorithm clearly outperforms the other two, and is thus the best candidate for applying it in PUIs.

7 Conclusion and Future Work

In this paper, we presented an overview of existing SPAs which can be used for predicting the next user action. We analyzed the parameters that have to be considered for the evaluation, especially focusing on the demands of a PUI. We found that the simple FxL algorithm performs best and is thus the prime candidate for the incorporation in PUIs. However, we saw that the prediction accuracy achievable with pure statistical approaches is limited to 40-60%. We assume that integrating task and context knowledge can improve these results. This leads to the problem how this task knowledge should be acquired. To solve this problem, we aim at combining modeled and learned knowledge. The PUI learns the taskmodel from observation, e.g., which actions are available and which context information corresponds to them. As the knowledge learnable by observation is limited, we enable the user to modify and extend the taskmodel, and to enrich it with constraints and interrelations between task model and context information. In the ideal case, such an enhanced taskmodel is provided by the application itself.

Acknowledgments
We would like to thank SAP Research Darmstadt for supporting our research in the AUGUR project.

References

[Albrecht *et al.*, 1998] David W. Albrecht, Ingrid Zukerman, and Anne E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8(1-2):5–47, 1998.

[Davison and Hirsh, 1998] Brian D. Davison and Haym Hirsh. Predicting Sequences of User Actions. In *Proceedings of AAAI-98/ICML-98 Workshop Predicting the Future*, pages 5–12. AAAI Press, 1998.

[Findlater and McGrenere, 2004] Leah Findlater and Joanna McGrenere. A comparison of static, adaptive, and adaptable menus. In *Proceedings of SIGCHI*, 2004.

[Gajos *et al.*, 2006] Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. Exploring the design space for adaptive graphical user interfaces. In *Proceedings of AVI '06*. ACM Press, 2006.

[Gopalratnam and Cook, 2007] Karthik Gopalratnam and Diane J. Cook. Online sequential prediction via incremental parsing: The active lezi algorithm. volume 22, pages 52–58, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[Gorniak and Poole, 2000] Peter Gorniak and David Poole. Predicting future user actions by observing unmodified applications. In *AAAI/IAAI*, 2000.

[Greenberg, 1988] Saul Greenberg. Using unix: collected traces of 168 users. Research report 88/333/45, 1988.

[Hartmann and Schreiber, 2007] Melanie Hartmann and Daniel Schreiber. A perl library of sequence prediction algorithms. "http://www.tk.informatik.tu-darmstadt.de/index.php?id=augur", 2007.

[Jacobs and Blockeel, 2002] N. Jacobs and H. Blockeel. Sequence prediction with mixed order Markov chains. In *Proceedings of the Belgian/Dutch Conference on Artificial Intelligence*, 2002.

[Künzer *et al.*, 2004] A. Künzer, F. Ohmann, and L. Schmidt. Antizipative modellierung des benutzerverhaltens mit hilfe von Aktionsvorhersage-Algorithmen. *MMI-Interaktiv*, (7), 2004.

[Linton *et al.*, 2000] Frank Linton, Deborah Joy, Hans-Peter Schaefer, and Andrew Charron. Owl: A recommender system for organization-wide learning. *Educational Technology & Society*, 3(1), 2000.

[Mitchell and Shneiderman, 1989] J. Mitchell and B. Shneiderman. Dynamic versus static menus: an exploratory comparison. *SIGCHI Bull.*, 20(4), 1989.

[Sears and Shneiderman, 1994] Andrew Sears and Ben Shneiderman. Split menus: effectively using selection frequency to organize menus. *ACM Trans. Comput.-Hum. Interact.*, 1(1):27–51, 1994.