Using feedback in adaptive and user-dependent one-step decision making

Abir-Beatrice Karami, Anthony Fleury

Using feedback in adaptive and user-dependent one-step decision making

Abir B. Karami
Mines Douai, IA, F-59508 Douai, France
Univ. Lille, F-59000, Lille, France
Email: abir.karami@mines-douai.fr

Anthony Fleury
Mines Douai, IA, F-59508 Douai, France
Univ. Lille, F-59000, Lille, France
Email: anthony.fleury@mines-douai.fr

Abstract

Several machine learning approaches are used to train systems and agents while exploiting users' feedback over the given service. For example, different semi-supervised approaches employ this kind of information in the learning process to guide the agent to a more adaptive and possibly personalized behavior. Whether for recommendation systems, companion robots or smart home assistance, the trained agent must face the challenges of adapting to different users (with different profiles, preferences, etc.), coping with dynamic environments (dynamic preferences, etc.) and scaling up with a minimal number of training examples. We are interested in this paper in one-step decision making for adaptive and user-dependent services using users' feedback. We focus on the quality of such services while dealing with ambiguities (noise) in the received feedback. We describe our problem and we concentrate on presenting a state of the art of possible methods that can be applied. We detail two algorithms that are based on existing approaches. We present comparative results by showing scaling and convergence analysis with clean and noisy simulated data.

1 Introduction

Nowadays, people expect new technologies in web services, recommendation systems, robotic assistance, smart assisted living, etc. to be adaptive and personalized to their own preferences.

It is important that adaptive systems (or agents) learn and update their knowledge and models continuously and in a natural way while offering their services. At the beginning, the interaction with such systems was unidirectional (a system service towards the user). However, in the last two decades, bidirectional architectures started to appear permitting users to express their level of satisfaction (feedback) back to the system allowing the latter to improve the quality of its service.

Learning an adaptive policy from user feedback is commonly used in the literature whether for sequential (long horizon planning) or one-step decision making frameworks. In sequential decision making a policy is calculated to maximize the long-term received rewards. Such problems are often represented by Markov Decision Processes (MDPs) [Howard, 1960] in which the MDP model and/or the policy are learned classically by Reinforcement Learning (RL) [Sutton and Barto, 1998] among other methods. Several authors take into consideration the user feedback in the RL process by considering them as shaping signals [Knox *et al.*, 2013], advice regarding the agent policy [Griffith *et al.*, 2013] or a simple evaluation of a selected action [Akrour *et al.*, 2012].

In one-step decision making a policy is calculated to maximize the immediate (one-step ahead) reward. Contrary to sequential decision making, no representation of the system dynamics is needed to calculate the policy. Such problems can be represented with Contextual Multi-Armed Bandits (CMAB) [Wang *et al.*, 2005; Langford and Zhang, 2007] (also called associative reinforcement learning [Strehl *et al.*, 2006]). User feedback is also employed in CMAB policy learning algorithms [Loftin *et al.*, 2014]. However, efficient proposed algorithms to solve CMAB problems have a high complexity and are time consuming [Agarwal *et al.*, 2014]. In addition, most proposed solutions do not represent negative feedback.

In this paper, we concentrate on adaptive one-step decision making using users feedback for agents in multi-users environments. We discuss about the capacity of such agents to make personalized decisions by representing users' profiles in the context set of attributes. It is expected from learning agents to adapt to dynamic change in their environments, including change in users' preferences. However, in this paper, we also discuss about the capacity of the agent to handle ambiguities in received feedback (communication errors, noise or missing information in the representation).

The approaches presented in this paper are based on detecting the relevance between the learned rewards (user x prefers watching cartoon media), the attributes representing the environment (rainy day) and the user profile (age < 10). Such relevance is an important key element for generalizing the learned adaptive behavior to new situations and unknown users and for decreasing the complexity of convergence to an optimal adaptive policy (decreasing the size of needed training data to reach an acceptable adaptive behavior).

In the following we will present a state of the art of existing methods for one-step decision making in Section 2. In Sec-

tion 3, we present our two proposed algorithms for learning an adaptive reward function by generalization and by using users' feedback. Then, we present, in Section 4, our results based on simulated training data to show scaling and performance analysis. Finally, we conclude with a brief discussion in Section 5.

2 Learning with feedback - state of the art

Our problem concerns an agent that makes a sequence of decisions during trials $\{t_1, t_2, \ldots, T\}$. At each trial, the agent is given a context represented by an attribute (feature) vector, or what is called a *state* in reinforcement learning. A context $x_t \in \mathcal{X}$ contains a set of attributes values representing the environment state and the current user profile. The current user is the one in interaction with the agent during the trial t, assuming that there is only one user during a trial. The agent is also given a set of possible actions $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$, similar for each trial. At each trial the agent selects an action and collects the feedback over the selected action in the form of a reward $r \in [-1, +1]$. Therefore, in t trials the agent register the following sequence of (context, action, reward) triples: $(x_1, a_1, r_1)(x_2, a_2, r_2) \ldots (x_t, a_t, r_t)$. We will also call that training examples and refer to them by $\mathcal{TE}$. These sequences or a processed version of them (through learning algorithms) are represented in a reward function $\mathcal{R} : x, a \rightarrow [-1, +1]$. In the next trial $(t + 1)$, the agent chooses the action that maximizes the immediate expected feedback reward knowing $\mathcal{R}$; $a_{t+1} = argmax_a \mathcal{R}_t(x_{t+1}, a), \forall a \in \mathcal{A}$.

We are interested in learning, with the least number of training examples, a generalized reward function that permits the agent to behave in an adaptive and personalized manner depending on the given context (that represent also the current user). We are also interested in the capacities of the learning algorithm to deal with ambiguities (noise) in users feedback. In other words, we are interested in the attitude of the learning algorithm and the performance of the resulted behavior if the agent receives the following two sequences $(x_1, a_1, r_1)(x_2, a_2, r_2)$, where $x_1 = x_2$ and $a_1 = a_2$ and $r_1 = -1 * r_2$.

The feedback received by the agent represent only the feedback over its chosen action. Feedback concerning the non-chosen actions remain unknown by the agent. Therefore, this problem can not be defined as a supervised learning problem. Other related approaches use semi-supervised learning, like active learning methods, where agents interactively query the user feedback on certain contexts that are determined as important in the learning process. However, such mechanisms are difficult to use in service agents scenarios (companion robots, smart homes, etc.) where the system is not in control of the current context nor of the current user.

In the following we present a brief state of the art while concentrating on methods related to those proposed and analyzed in this paper and that we chose a priori as interesting solutions regarding the described problem and its properties.

Using supervised (or semi-supervised) learning algorithms to learn decision rules from training sets is a known problem in the literature. The ID3 algorithm [Quinlan, 1986] is a very known supervised learning algorithm that learns the smallest decision tree that represents all the training examples. The ID3 algorithm is based on selecting the attribute with the highest information gain and splits the data (the decision tree) to a number of branches equal to the number of values of this attribute. The algorithm iterates until each branch of the decision tree represents only one class of the training examples. An important drawback of this algorithm is the fact that it can over-fit the training data. Pruning methods [Mingers, 1989] can be used to deal with this problem. We present in this paper a slightly modified version of the ID3 algorithm that learns a generalized reward function by selecting the important attributes with the highest information gain.

The version space [Russell and Norvig, 2003] is a supervised learning approach used for binary classification. It searches for a predefined space of hypotheses. While being presented with the training examples, the candidate elimination algorithm works on two sets of hypotheses, the most specific one and the most general one. If both sets reaches a similar representation, the algorithm reaches a solution. The major drawback of this algorithm is its inability to deal with noise, which means that any detected contradiction can cause the version space to fail in the learning process. In a previous work [Karami *et al.*, 2016b], a generalized version of the classic version space algorithm was proposed. This version is inspired by the idea of specializing and generalizing the set of rules representing the training examples. It is also based on the idea of detecting important attributes which helps in generalizing the learned rules in a way that represents not only the training examples but also other examples that were not yet evaluated. We also present the Generalized Version Space algorithm in this paper.

3 Methods for one-step adaptive and personalized decision making using feedback

We detail in this section, a method that is based on the ID3 decision tree induction algorithm and another based on the concept of generalization and specialization of the version space candidate elimination algorithm. Both methods benefit from the concept of detecting important attributes for each possible action. By definition, an important attribute (whether a user profile attribute or another environmental attribute) is one that has a value that affects the Feedback-value-Direction (FD) (*i.e.* the user feedback r is positive or negative). We denote $\mathcal{IAT}_a$ the set of important attributes concerning the action a. Attributes for which the values are not important are generalized to any value (*) in the generalized reward function $\mathcal{R}$ of the concerned action a.

In the generalized version space algorithm (GVS) (Algorithm 1), $\mathcal{R}$ is initiated with a very general rule (one reward with all the attributes represented by a * for all the actions and $r = 0$). However, while receiving new sequences, the algorithm tries to detect important attributes by detecting contradictions between a new sequence and the rules in $\mathcal{R}$. If an important attribute is detected, a specialization phase is required in which all the past registered sequences are used to add new specified rules to $\mathcal{R}$ with respect to the important attributes values.

The Reward Learner Iterative Dichotomiser 3 (RLID3) algorithm (Algorithm 2) contrary to the GVS algorithm uses the whole set of past registered sequences with the new received sequences to conduct the rules of $\mathcal{R}$.

Detecting important attributes allows to learn the reward function faster, minimize the needed number of training examples and generalize the learned function to represent unknown contexts (unknown users with new profiles and/or new environmental settings).

3.1 The Generalized Version Space algorithm

The used mechanism in this algorithm is inspired from the version space generalizing and specializing techniques. The algorithm input is the new received training example te_a, the current reward function regarding the same action $\mathcal{R}_a$, the set of important attribute $\mathcal{IAT}_a$, and the set of Treated Training Examples $\mathcal{TTE}_a$. Those sets are initialized as empty sets before the reception of the first training example and then updated during the treatment of each new received one. The outputs are the modified versions of $\mathcal{R}_a, \mathcal{IAT}_a$, and $\mathcal{TTE}_a$ after treating te_a. This algorithm is called at each reception of a new training example and only subsets and functions that are related to the action a are used and potentially modified.

The algorithm backs up all received and treated training examples (past sequences) (line 17), so there is no loss of information because of the generalization. The $\mathcal{TTE}_a$ set is continuously used in the process of detecting important attributes.

Algorithm 1 The Generalized Version Space (GVS) algorithm

1: **INPUT** $te_a, \mathcal{R}_a, \mathcal{IAT}_a, \mathcal{TTE}_a$.
2: **Output** $\mathcal{R}_a, \mathcal{IAT}_a, \mathcal{TTE}_a$
3: **for all** $r_a \in \mathcal{R}_a$ **do**
4: **if** (r_a includes te_a) and (r_a not same $\mathcal{FD}$ as te_a) **then**
5: Put in $related\_\mathcal{TTE}_a$ all related to r_a.
6: **for all** $rtte \in related\_\mathcal{TTE}_a$ **do**
7: **for all** $at \in \mathcal{AT}$ **do**
8: **if** (at in $rtte! = at$ in te_a) **then**
9: **if** ($\mathcal{IAT}_a$ is empty) **then**
10: Add at to $\mathcal{IAT}_a$.
11: **else**
12: **if** (iat !pertinent with $\mathcal{IAT}_a$) **then**
13: Add at to $\mathcal{IAT}_a$.
14: Add te_a to $\mathcal{R}_a$.
15: Generalize all non important attributes in $\mathcal{R}_a$.
16: Check $\mathcal{R}_a$ for redundancies.
17: Add te_a to the backup set $\mathcal{TTE}_a$.

The algorithm searches for contradictions between te_a and $\mathcal{R}_a$ to detect important attributes (line 4). If a reward r_a includes te_a and if both does not have the same $\mathcal{FD}$ (one has a positive or null reward and the other has a negative reward), then a contradiction is detected. The inclusion of a te_a by an r_a is guaranteed if the attributes values are equal or general enough in r_a (* for example) to represent the values of the attributes in te_a. In our simulated experiments, we considered null feedback (0) as positive feedback.

If a contradiction is detected, the algorithm tests if there is an important attribute concerning the action a (lines 5:13). To do that, first, a non empty set of related treated training examples is set to $related_{\mathcal{TTE}_a}$ (line 5). We consider an $rtte \in related_{\mathcal{TTE}_a}$ related to r_a if they both concern the same action, they have the same $\mathcal{FD}$ and the attributes values in $rtte$ are equal or represented by the attributes values in r_a. Secondly, the algorithm searches for attributes in $rtte$ with different values than te_a (line 8). If found, the concerned attributes are added to the set of important attributes for action a ($\mathcal{IAT}_a$) (line 10).

When dealing with noisy data, it is possible that the algorithm marks some attributes as important when they are actually not. To overcome this problem, we added a function that checks for falsely detected important attributes (Lines 12:13). In more detail, an important attribute iat is pertinent and confirmed if the following condition is true for each $r_a \in \mathcal{R}_a$: there exist at least one other reward in $\mathcal{R}_a$ where: for all other important attributes $iat_i \neq iat$ the value of the attribute in r_{a_i} is equal to its value in r_a and both rewards have different $\mathcal{FD}$.

If any change occurs to the set of important attributes, the algorithm updates the set of learned rewards by generalizing all attributes that are not set as important (setting their values to *) (line 15). Moreover, after generalizing all non important attributes, the algorithm checks the set of rewards $\mathcal{R}_a$ for any redundant rewards (line 16). The step of checking for redundancies insures that at each reception of a new te_a there is at most one $r_a \in \mathcal{R}_a$ that could match it. In this case lines (6:15) are applied. If no match exists, te_a is simply added as a new reward in $\mathcal{R}_a$ (line 14).

3.2 Decision tree induction based algorithm

Decision trees are known to be interpretable and compact. More importantly, algorithms like ID3 handles irrelevant attributes which helps in decreasing the complexity of needed examples to learn a representative decision tree. Algorithm 2 presents the Reward Learner Iterative Dichotomiser 3 algorithm (RLID3). The RLID3 is similar to the ID3 algorithm, however, the output is represented as a reward function $\mathcal{R}$ instead of a node based decision tree.

Algorithm 2 input is the set of all training examples $\mathcal{TE}_a$, the current reward function regarding the same action $\mathcal{R}_a$, and the set of selected attributes which helps in guiding the algorithm through its recursive mechanism. The last two sets are initialized to an empty sets at the beginning of the algorithm. The output is the modified versions of $\mathcal{R}_a$ and $\mathcal{IAT}_a$, after treating all the training examples $\mathcal{TE}_a$. The algorithm is based on the same Information Gain (IG) and Entropy (H) as the classic $ID3$ algorithm. The selected best attribute is the one that gives the higher IG of all attributes and is added to the $selected\_AT$ set and important attributes $\mathcal{IAT}_a$ set (lines 3:5).

$$IG(\mathcal{AT}, \mathcal{TE}) = H(\mathcal{TE}) - \sum_{k \in K} p(k)H(k), \quad \text{where}$$

$$H(\mathcal{TE}) = -\sum_{c \in C} p(c) log_2 p(c)$$

C is the set of classes in $\mathcal{TE}$, here in our case we have 2 classes (the positive and the negative examples). $H(\mathcal{TE})$ is the Entropy of the set of training example. K is the subsets created by splitting $\mathcal{TE}$ by the attribute $\mathcal{AT}$. $p(c)$ and $p(k)$ represent the proportion of the number of training examples in the subsets c and k respectively to the total number of examples in $\mathcal{TE}$. $H(k)$ is the entropy of subset k.

If a best attribute was found, then the algorithm splits the training examples into a number of subsets equal to the possible values of the best attribute. Each subset contains the examples that correspond to the best attribute value (line 7). Then, each of these subsets is tested, if all the examples of the subset are positive (line 8) then a new positive reward is created with the corresponding value of the $selected\_\mathcal{AT}$ and the best attribute $best\_at$ is removed from the $selected\_\mathcal{AT}$ (lines 9:12). The same process is followed, however by creating a negative reward, if all the examples of the subset are negative (line 13:17). If the examples of the subset consists of a mix of positive and negative examples, the algorithm is called recursively to select a new best attribute using the information gain function, and so on until no more training examples to consider.

Algorithm 2 RLID3 algorithm

```
1: INPUT TE_a, R_a, selected_AT.
2: Output R_a, IAT_a.
3: Select the best attribute and assign it to best_at.
4: Add best_at to the list of selected_AT.
5: Add best_at to the list of IAT_a.
6: if (best_at! = null) then
7:     for all values of best_at do
8:         if (all examples are positive) then
9:             New r_a with values of selected_AT.
10:            Set the reward of r_a to +1.
11:            Add r_a to R_a.
12:            Remove best_at from selected_AT.
13:        else if (all examples are negative) then
14:            New r_a with values of selected_AT.
15:            Set the reward of r_a to −1.
16:            Add r_a to R_a.
17:            Remove best_at from selected_AT.
18:        else
19:            RLID3(TE_a, R_a, selected_AT).
20:            Remove best_at from selected_AT.
```

4 Simulated experimental results

We present in this section some scaling and performance analysis using the GVS and RLID3 algorithms compared with a simple memorizing method without generalization (where the $\mathcal{R}$ is equivalent to the training examples). Those results are based on simulated training examples. In the following we describe the procedure that we followed to simulate the training examples, the procedure of the experiment and the results of our analysis.

4.1 Parameters and simulation of training examples:

In our simulations we used the following parameters: the number of attributes representing the context including user profile, the number of actions, the number of possible values for each attribute, and the number of important attributes for each action. The parameters values are detailed later. Some of them were fixed manually and others were decided randomly using predefined range of values depending on the experiment.

A training example consists of a context, an action and a feedback. We simulated contexts by giving a random value for each attribute respecting the possible number of values for each of them (defined in the parameters). To simulate the users feedback we used some predefined rules of preference. The predefined rules were generated randomly based on the number of important attributes for each action. The selected important attributes used in the simulations were later compared with the detected important attributes by the GVS and RLID3 algorithms.

Each simulated training example was based on a simulated context. The action is chosen in a way that maximizes the immediate reward (see Section 2) over the current version of the reward function $\mathcal{R} = \bigcup_{\forall a \in \mathcal{A}} \mathcal{R}_a$. Then, the predefined rules of preference are used to simulate the user feedback over the chosen action. A predefined rule concerning the action a is applicable in a context if the important attributes values of the rule have the same values in the concerned context.

To balance between exploitation and exploration during action selection, we followed the epsilon greedy method with $\epsilon = 10$. This means that with 90% chance the algorithms chose the best action the maximizes the immediate reward, however, with 10% chance a random action is chosen. Such behavior helps in exploring eventually a global optima instead of exploiting a local optima [Langford and Zhang, 2008].

4.2 Scaling Analysis

Table 1 shows some scaling analysis. We modified the parameters values and studied the effects on needed time to treat all training examples and the number of detected important attributes. During this experiment we simulated training examples with noise (ambiguity in users feedback). With a 3% probability while simulating each training example the user feedback was reversed (negative if positive, positive if negative).

The parameters presented in Table 1 are:

- the number of attributes $|\mathcal{AT}|$,
- the number of possible values for each attribute $|v\mathcal{AT}|$ (for example in the first experiment in line 1, $|v\mathcal{AT}|$ was decided randomly between 2 and 4 possible values for each attribute),
- the calculated number of possible contexts $|\mathcal{S}|$ using $|\mathcal{AT}|$ and $|v\mathcal{AT}|$,
- the number of actions $|\mathcal{A}|$,
- the number of important attributes per action $|\mathcal{IAT}|$ (for example in the first experiment presented in line 1,

	Problem size						Time			Detected $\|IAT\|$	
	$\|\mathcal{AT}\|$	$\|v\mathcal{AT}\|$	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	$\|\mathcal{IAT}\|$	$\|\mathcal{TE}\|$	**Simple**	**RLID3**	**GVS**	**RLID3**	**GVS**
1	7	$\in$ [2,4]/$\mathcal{AT}$	432	5	$\in$ [1,3]/a = 11	1000	<1s	<1s	<1s	11/11	10/11
2	7	$\in$ [2,6]/$\mathcal{AT}$	10368	5	$\in$ [1,3]/a = 8	5000	<1s	<1s	<1s	8(+1)/8	8/8
3	10	$\in$ [2,4]/$\mathcal{AT}$	27648	5	$\in$ [1,3]/a = 12	5000	<1s	<5s	1min10s	11/12	12/12
4	10	$\in$ [2,4]/$\mathcal{AT}$	49152	10	$\in$ [1,3]/a = 22	5000	<1s	<5s	2min56s	19(+6)/22	11/22
5	10	$\in$ [2,6]/$\mathcal{AT}$	225000	10	$\in$ [1,3]/a = 21	5000	<1s	<10s	6min45s	19(+1)/22	11(+7)/22
6	7	$\in$ [2,6]/$\mathcal{AT}$	9600	5	$\in$ [1,6]/a = 22	5000	<1s	<5s	<10s	22(+5)/22	22/22
7	10	$\in$ [2,6]/$\mathcal{AT}$	216000	10	$\in$ [1,6]/a = 34	5000	<5s	<10s	14min40s	29(+17)/34	23/34

Table 1: Scalability analysis - time versus problem size and efficiency in detecting important attributes on noisy data set (3% ambiguity).

$|\mathcal{IAT}|$ was decided randomly between 1 and 3 important attributes per action), the table also shows the total effective number of important attributes over the set of actions (11 in the first experiment),

- the number of training examples $|\mathcal{TE}|$.

The time columns show the needed time to treat all the training examples using the simple memorizing algorithm, the GVS and the RLID3 algorithms. The Detected $|\mathcal{IAT}|$ columns show the number of detected attributes by the GVS and the RLID3 algorithms over the real number of important attributes. The second experiment for example (described in line 2), shows that the RLID3 algorithm detected the total of 8 important attributes, however, it also detected another attribute that was not really assigned as important in the simulations. It is important to mention that the calculated time included the time of simulating the training examples in addition to model creation and display time.

The results show that the GVS algorithm is time consuming (the calculation time is exponential to the number of contexts $|\mathcal{S}|$). The detection of the important attributes, however, is not optimal by both algorithms for problems with important sizes (>50000 contexts). Nevertheless, agents can behave in an acceptable manner even if some important attributes were not detected. We show some performance analysis in the following of this section.

4.3 Performance results:

In the following experiments, we fixed $|\mathcal{AT}| = 8$, $|v\mathcal{AT}| \in [2, 4]$, $|\mathcal{A}| = 3$, $|\mathcal{IAT}| \in [1, 3]$ and $|\mathcal{TE}| = 100000$. To evaluate the evolution of the performance with time, we treated the training examples by epochs of 100 at each time. For each epoch, the procedure that we followed for the evaluation is as follows:

1. simulate 100 training examples by selecting actions based on the learned $\mathcal{R}$,
2. evaluate the chosen actions during the epoch by counting the number of training examples including negative feedback (referred to as *number of negative actions*),
3. learn/update the reward function $\mathcal{R}$ using the algorithms over the 100 simulated training examples (the GVS algorithm treats only the new 100 simulated training examples while the RLID3 algorithms treats the complete sets used in previous epochs in addition to the new 100 ones).

Epochs are repeated until all training examples are treated ($100000/100 = 1000$ epochs in our experiments).

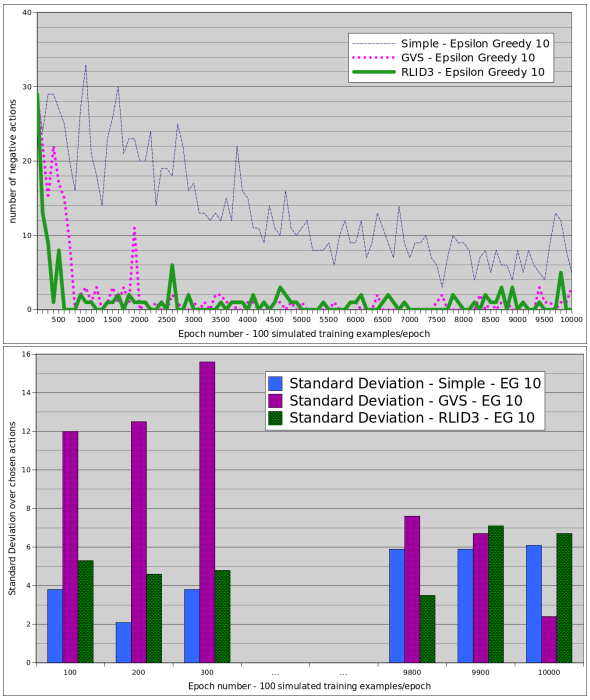

Figure 1: Simulated results comparing the GVS and RLID3 algorithms with a simple method using an epsilon greedy mechanism with $\epsilon = 10$ in the action selection process. (**a**) The number of negative actions in each epoch. (**b**) The standard deviation between the number of times each action was used during an epoch.

Figure 1 shows simulated results using the explained procedure with an exact number of contexts $|\mathcal{S}| = 3072$ and an exact total number of important attributes of 12. The figure shows results using the GVS and RLID3 algorithms compared with the simple method. All 3 algorithms followed the epsilon greedy mechanism with $\epsilon = 10$. The upper part of the figure shows the number of negative actions in each epoch. We notice that the RLID3 and the GVS algorithms needed less than 300 and 800 training examples respectively to perform with less than 5 negative actions per epoch. However, the simple memorizing method without generalization did not

reach the same performance before treating more than 8000 training examples.

Regarding the detection of important attributes, both GVS and RLID3 algorithms were able to detect the total of 12 important attributes.

The lower part of the figure shows the standard deviation for the number of times each of the 5 actions were used. The value of this standard deviation demonstrates that different actions were selected by the algorithms (and not always the same action). In the first few epochs the GVS generates a high standard deviation because it selected the same "with low risk" action most of the time. However, through learning and exploring (epoch greedy) the standard deviation decreases because other actions were found to be more advantageous.

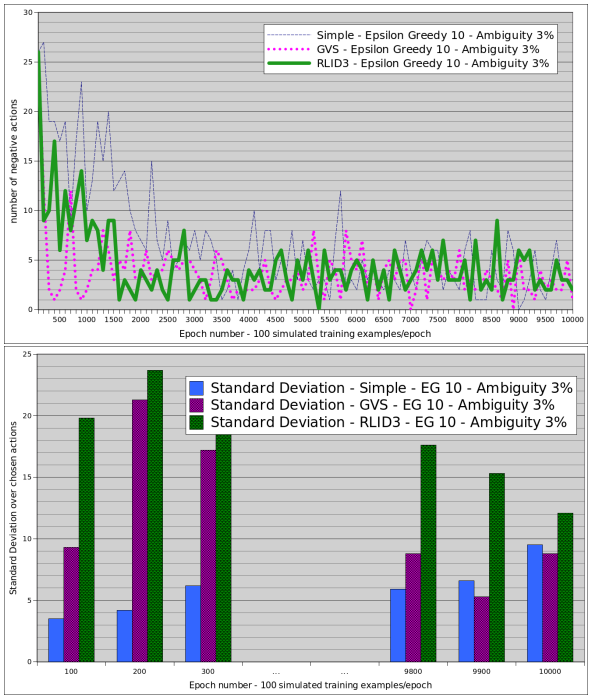

Figure 2: Simulated results comparing the GVS and RLID3 algorithms with a simple method using an epsilon greedy mechanisme with $\epsilon = 10$ in the action selection process and with an ambigtuiy of 3% in users feedback. (**a**) The number of negative actions in each epoch. (**b**) The standard deviation between the number of times each action was used during an epoch.

Results of a second simulated experiment are shown in Figure 2. In this experiment there was an effective total of 576 contexts and 7 important attributes. We tested in this experiment the capacity of the algorithm to deal with noise (ambiguity in users feedback). During the simulation of training examples we reversed the user feedback (negative if positive, positive if negative) with a probability of 3%. The epsilon greedy with $\epsilon = 10$ was also respected by all algorithms in this experiment. Results show that both GVS and RLID3 algorithms converge to around 5 negative actions per epoch after treating more than 1200 training examples. However, the GVS gave some better results (by exploiting a local optima) during the first 1000 examples. As expected, both algorithms did not reach an optimal convergence because the predefined rules of preferences can not be regenerated through examples with ambiguous users feedback. The standard deviation analysis shows that even with ambiguity in users feedback both algorithms choose interesting actions instead of repeating the least risky ones.

These results prove the interest of detecting important attributes for a better and faster adaptation. We notice that the RLID3 algorithm outperforms the GVS on clean data and the GVS outperforms the RLID3 algorithm on noisy data. However, it is clear from the scaling analysis that the GVS algorithm is time consuming and can reach intractability in large contexts (with a state space of a million states and an important percentage of important attributes for each action, up to 75% for example). On the other hand, RLID3 showed a very reasonable calculation time with less performance in noisy environment. We are actually studying the capacity of ID3 pruning mechanisms to deal with the problem of noise and ambiguity in users feedback.

5 Conclusion and Future Work

We proposed in this paper two methods to guide a learning agent to an adaptive and personalized behavior in multi-user environments. Our proposed methods are based on existing supervised learning and classification approaches, even though the definition of our problem shows that it falls in the semi-supervised learning category. Therefore, the presented state of the art was focused on the types of methods that we chose a priori as interesting solutions regarding the described problem and its properties. Other classification algorithms and semi-supervised approaches can be interesting to compare with, which is the subject of a future work.

In previous work, a complexity and convergence analysis is presented for the GVS algorithm, in addition to a real experiment that proved the applicability of this algorithm for an adaptive and personalized behavior of a companion robot [Karami *et al.*, 2016b]. A study of user satisfaction was also presented regarding the personalized behavior of the companion robot. We intend to test the RLID3 algorithm in a smart home environment that adapts for the comfort of its occupants while using their feedback over the automated control of the smart home [Karami *et al.*, 2016a].

Potentially, it seems interesting to study the source of ambiguities in user feedback. The most classic reason is the noise in the received data (voice recognition for vocal feedback, image processing for facial nod, etc.). However, ambiguities can be caused because of lack of information in the representation model. For example, a user might change his/her preference in a rainy day, which informs us of the need of adding weather forecast as an attribute. Integrating to the learning algorithm an automatic mechanism to detect missing information in the representation model can be interesting, however, it should include a complete study of how to transfer a complete and connected set of contexts to a human expert to facilitate the identification of the missing attribute(s).

References

[Agarwal *et al.*, 2014] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert E Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. *arXiv preprint arXiv:1402.0555*, 2014.

[Akrour *et al.*, 2012] Riad Akrour, Marc Schoenauer, and Michèle Sebag. APRIL: Active Preference-learning based Reinforcement Learning. In P. Flach et al., editor, *ECML PKDD 2012*, volume 7524 of *LNCS*, pages 116–131, Bristol, Royaume-Uni, September 2012. Springer Verlag.

[Griffith *et al.*, 2013] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2625–2633, 2013.

[Howard, 1960] Ronald A Howard. Dynamic programming and markov processes.. 1960.

[Karami *et al.*, 2016a] Abir B. Karami, Anthony Fleury, Jacques Boonaert, and Stéphane Lecoeuche. User in the loop: Adaptive smart homes exploiting user feedback state of the art and future directions. *Information*, in press, 2016.

[Karami *et al.*, 2016b] Abir B. Karami, Karim Sehaba, and Benoit Encelle. Adaptive artificial companions learning from users feedback. *Adaptive Behavior*, 24(2):69–86, 2016.

[Knox *et al.*, 2013] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Training a robot via human feedback: A case study. In *Social Robotics*, pages 460–470. Springer, 2013.

[Langford and Zhang, 2007] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 817–824. Curran Associates, Inc., 2007.

[Langford and Zhang, 2008] John Langford and Tong Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*, pages 817–824, 2008.

[Loftin *et al.*, 2014] Robert Loftin, James MacGlashan, Bei Peng, Matthew E Taylor, Michael L Littman, Jeff Huang, and David L Roberts. A strategy-aware technique for learning behaviors from discrete human feedback. In *Proc. of AAAI*, 2014.

[Mingers, 1989] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2):227–243, 1989.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Russell and Norvig, 2003] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[Strehl *et al.*, 2006] Alexander L Strehl, Chris Mesterharm, Michael L Littman, and Haym Hirsh. Experience-efficient learning in associative bandit problems. In *Proceedings of the 23rd international conference on Machine learning*, pages 889–896. ACM, 2006.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge, 1998.

[Wang *et al.*, 2005] Chih-Chun Wang, Sanjeev R Kulkarni, and H Vincent Poor. Bandit problems with side observations. *Automatic Control, IEEE Transactions on*, 50(3):338–355, 2005.