Learning Users' Assistance Requirements with WATSON

Silvia Schiaffino
ISISTAN Research Institute – Fac. Cs. Exactas – UNICEN
Paraje Arroyo Seco, Campus Universitario – Tandil, Argentina
sschia@exa.unicen.edu.ar
Also CONICET

Abstract. Interface agents are computer programs that provide personalized assistance to users with their computer-based tasks. The interface agents developed so far have focused their attention on learning a user's preferences in a given application domain and on assisting him according to them. However, in order to personalize the interaction with users, interface agents should also learn how to best interact with each user and how to provide them assistance of the right sort at the right time. Particularly, an interface agent has to determine when the user wants a suggestion to solve a problem, when he requires only a warning about it, when he wants the agent to execute an action to deal with the problem and when he wants the agent to do just nothing. In this work we propose a learning algorithm, named *WATSON*, to tackle this problem. The *WATSON* algorithm enables an interface agent to adapt its behavior and its interaction with a user to the user's assistance requirements.

1. Introduction

Interface agents have become a technology widely used to provide personalized assistance to users with their computer-based tasks. Interface agents are computer programs that have the ability of learning a user's preferences, and helping him to deal with one or more computer applications, reducing in this way the user's workload.

A commonly used metaphor for understanding interface agents paradigm is comparing them to a human secretary or personal assistant who is collaborating with the user in the same work environment [9]. Initially, a personal assistant is not very familiar with the habits and preferences of her employer and may not be very helpful. However, the assistant becomes gradually more effective and competent as she acquires knowledge about him. This knowledge can be acquired by observing how the employer performs tasks, by asking the employer for information, by receiving explicit instructions and feedback from him, or by learning from the experience of other assistants. Then, the assistant can perform tasks that were initially performed by the employer, she can suggest him the execution of tasks, she can notify the employer about situations interesting for him and warn him about problems that may arise. In the same way, an interface agent can become more competent as it interacts with a user and learns about him.

The many interface agents that have been developed focus their attention on learning a user's preferences in a given application domain and on assisting the user according to them. However, interface agent developers have paid little attention to some key issues when assisting a user: how to best interact with each user and how to provide them assistance of the right sort at the right time.

In this context, Horvitz [7] has pointed out some problems with the use of interface agents: poor guessing about the goals and needs of users, inadequate consideration of the costs and benefits of each agent action and poor timing of agent actions. In addition to these problems, we have identified a specific problem that has to be solved in order to personalize and improve the interaction between an interface agent and a user. When an interface agent detects a problem situation or situation of interest, it has to decide between warning the user about the problem and let him decide how to deal with it, suggesting him how to solve the problem or solving it on the user's behalf. This decision will be mainly influenced by the knowledge the agent has to make a suggestion, since if it does not know what to suggest or what to do, it will merely warn the user about the problem. However, although the agent probably knows what to suggest, it has to learn if the user wants it to suggest him something or not in that particular problem situation, or if he does not even want to be warned. In order to take the best decision, the agent must determine when the user wants a suggestion to solve a problem, when he requires only a warning about it and when he wants the agent to solve the problem.

Consider, for example, an interface agent that helps a user with the organization of his agenda. When a conflict between two activities arises, the agent can warn the user about it, it can also suggest the user how to solve it, i.e. suggest which event(s) the user should reschedule, or it can

reschedule the events on behalf of the user. However, the user does not handle all the conflicts in the same way. He wants the agent's help in some situations but he wants to handle some other conflicts by himself. Thus, an agent should learn when and how the user wants to be assisted in order to personalize its interaction with the user and become then more competent and usable.

In this work we propose a learning algorithm, named *WATSON*, to tackle the problem presented before. The learning algorithm uses association rules to determine when the user wants a given type of assistance. *WATSON* enables an interface agent to adapt its behavior and its interaction with a user to the user's assistance requirements.

This work is organized as follows. Section 2 presents our proposed learning algorithm. Section 3 reports some experimental results. Section 4 describes some related work. Finally, Section 5 presents our conclusions and future work.

2. WATSON Learning Algorithm

The goal of our algorithm is learning which assistance action the user expects from an interface agent in each problem situation or situation of interest that may arise. The following subsections describe in detail our algorithm.

2.1. Inputs and Outputs

The input for our learning algorithm is a set of user-agent interaction experiences in a given application domain. An interaction experience *Ex* is described by four arguments: a problem situation *P*, the assistance action *AA* the agent executed to deal with the problem, the user feedback *UF* obtained after assisting the user and an evaluation *E* of the assistance experience (success or failure). Each problem situation *P* is described by a set of features and the values these features take, $P=\{(feature_i,value_i)\}$. An assistance action may be a suggestion, a warning or an action. A suggestion is described by the suggestion the agent has made, the problem originating it and a justification of the proposed solution. Similarly, an action is described by a set of parameters describing the action performed (these parameters are application dependent), the problem originating it and a justification of the decision of executing that action. A warning is simply described by the problem situation the user is being warned about. The user feedback may be explicit, if the user explicitly evaluates the agent's actions, or implicit if the agent has to obtain it from the user's actions. In turn, the user feedback can be positive or negative. It is positive if the user accepts the assistance provided by the agent, i.e. if the assistance action executed by the agent was the one the user expected. Otherwise, the feedback is negative. According to this feedback, the experience can be evaluated as a success or as a failure.

The output of our algorithm is a set of facts establishing which assistance action the user requires in a given problem situation. These facts may adopt one of the following forms: “in problem situation *P* the user requires a warning *W*”, “in problem situation *P* the user requires a suggestion *S*”, “in situation *P* the user wants the agent to execute action *A*” or “the user does not want assistance (in situation *P*)”. Each fact *F* is accompanied by a certainty degree *Cer(F)*, which indicates how certain the agent, is about this fact. The following sections describe how we obtain these facts from the set of user-agent interaction experiences.

2.2. WATSON Overview

The algorithm uses the information contained in a set of user-agent interaction experiences to formulate some hypotheses about the user's assistance requirements. A hypothesis expresses the agent's belief that the user requires a certain type of assistance in a given problem situation. A hypothesis *H* expresses that whenever problem situation *P* occurs, the user will require assistance action *AA* with a certainty degree of *Cer(H)*. Then, if a hypothesis is highly supported, i.e. if its certainty degree is greater than a threshold value δ, it is turned into a fact.

In this work we propose the utilization of association rules to formulate hypotheses from a set of user-agent interaction experiences. Association rules imply an association relationship among a set of items in a given domain. In our domain, they can be used to determine relationships among problem situations and the assistance actions required by users.

The association rules generated from the set of user-agent interaction experiences are post-processed in order to derive useful hypotheses from them. This post-processing includes detecting the most interesting rules from the generated ones, eliminating redundant and insignificant rules and summarizing the information in order to formulate the hypotheses more easily. The certainty degree of a hypothesis is computed using metrics used in association rule mining, as we will see later. Algorithm 1 shows the main steps our algorithm involves. The following sections explain in detail how we perform each of them.

INPUT: A set Ex of user-agent interaction experiences $Ex_i=<P_i, AA_i, UF_i, E_i>$ (where P: situation, AA: assistance action, UF: user feedback, E: evaluation)
OUTPUT: A set F of facts and a set H of hypotheses representing the user's assistance requirements

1. $F \leftarrow \varnothing$
2. $H \leftarrow \varnothing$
3. Generate a set of association rules AR (using Apriori algorithm for example) from Ex
4. $AR_1 \leftarrow$ Filter out uninteresting rules from AR
5. $AR_2 \leftarrow$ Eliminate redundant and insignificant rules from AR_1
6. $H \leftarrow$ Transform rules in AR_4 into hypotheses (just notation transformation)
9. FOR (i=1 to size of H){
   10. $Cer(H_i) \leftarrow$ compute certainty degree of H_i
   11. IF ($Cer(H_i) \geq \delta$)
      12. $F \leftarrow F \cup H_i$ }

Algorithm 1. *WATSON* Overview

2.3. Mining Association Rules from User-Agent Interaction Experiences

An association rule is a rule that implies certain association relationships among a set of objects in a database, such as occur together or one implies the other. Association discovery finds rules about items that appear together in an event (called transactions), such as a purchase transaction or a user-agent interaction experience. Association rule mining is commonly stated as follows [1]: Let $I=\{i_1,...,i_n\}$ be a set of items, and D be a set of data cases. Each data case consists of a subset of items in I. An association rule is an implication of the form $X \rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap = \varnothing$. X is the antecedent of the rule and Y is the consequent. The support of a rule $X \rightarrow Y$ is the probability of attributes (or attribute sets) X and Y occurring together in the same transaction. The rule has support s in D if s% of the data case in D contains $X \cap Y$. If there are n total transactions in the database, and X and Y occur together in m of them, then the support of the rule $X \rightarrow Y$ is m/n. The rule $X \rightarrow Y$ holds in D with confidence c if c\% of data cases in D that contain X also contain Y. The confidence of rule $X \rightarrow Y$ is defined as the probability of occurrence of X and Y together in all transactions in which X already occurs. If there are s transactions in which X occurs, and in exactly t of them X and Y occur together, then the confidence of the rule is t/s.

An example of an association rule is: "30% of transactions in a supermarket that contain beer also contain diapers; 2% of all transactions contain both items." In this example, 30% is the confidence of the rule and 2% the support of the rule. Given a transaction database D, the problem of mining association rules is to find all association rules that satisfy: minimum support (called minsup) and minimum confidence (called minconf). Minsup is an input parameter to the algorithm for generating association rules. It defines the support threshold, and rules that have support greater than minsup are the only ones generated. Minconf is an input parameter that defines the minimum level of confidence that a rule must possess.

There has been a lot of research in the area of Association Rules, and as a result there are various algorithms to discover association rules in a database. The most popular is the Apriori algorithm [1], which is the one we use to find our association rules. In this work we are not concerned about how association rules are generated. If readers want more information about association rule mining can read [1], [10], [2].

2.4. Post-processing of rules

In our work, we are interested in some particular association rules generated from a set of user-agent interaction experiences. The rules we are interested in are those association rules of the form *"problem description, assistance action → user feedback, evaluation"* having appropriate support and confidence values. Other combinations of items are irrelevant since we are trying to discover which problem situation-assistance actions pairs received a positive user feedback and were evaluated in consequence as a success.

We can use an intuitive approach [5] to select those rules we are interested in. Relevant (and also irrelevant) classes of rules can be specified with templates. Templates describe a set of rules by specifying which attributes occur in the antecedent and which attributes occur in the consequent. A template is an expression of the form: $A_1,...,A_k \rightarrow A_{k+1},..,A_n$, where each A_i is an attribute name, a class name, or an expression C+ and C*, which correspond to one or more and zero or more instances of the class C, respectively. A rule $B_1,...,B_h \rightarrow B_{h+1},...,B_m$ matches the pattern if the rule can be considered to be an instance of the pattern. Our agents will be interested in those rules where the Problem Description and the Assistance Action are on the left side and the User Feedback and the Evaluation are on the right side.

Once we have filtered those rules that were not interesting for us, we will still have many rules to process, some of them redundant or insignificant. We can then use a technique that removes those redundant and insignificant associations and then finds a special subset of the unpruned associations to form a summary of the discovered rules [8]. Many discovered associations are redundant or minor variations of others. Thus, those spurious and insignificant rules should be removed. For example, consider the following rules:

R1: EventType=doctor, warning → failure, askforsuggestion [sup:60%;conf:90%]
R2: EventType=doctor, Priority=high, warning → failure, askforsuggestion [sup:40%,conf=91%]

If we know R1, then R2 is insignificant because it gives little extra information. It thus should be pruned. R1 is more general and simple.

The remaining rules are the ones we consider to build hypotheses. The certainty degree of a hypothesis is computed using the support associated to the rule originating it.

3. Experimental Results

In order to evaluate the performance of our learning algorithm we used one of the metrics defined in [3]. The *precision metric* measures an interface agent's ability to accurately provide assistance to a user. We define our precision metric as shown in Equation 1.

$$M_{precision} = \frac{\text{number of correct assistance actions}}{\text{number of assistance actions}} \quad (1)$$

The precision metric is used to evaluate the performance of an interface agent when it has to decide among a warning, a suggestion or an action. In this case, for each problem situation, we compare the number of correct assistance actions against the total number of assistance actions the agent has executed. An assistance action is correct if it is the one expected by the user in a given problem situation. The user's feedback tells us whether an assistance action is correct or not.

We tested our algorithm with a set of 20 users of an agenda system. For this purpose, we incorporated the *WATSON* algorithm into an agent that provided assistance to these users. We compared the assistance capability of this agent with *WATSON* and without *WATSON*. We obtained the user feedback after each assistance action executed by each agent and we classified the assistance experience as a success or a failure. We used the WEKA[1] package to implement the process of association rules finding. The graph in Figure 1 plots the evolution of the precision metric both for an agent using *WATSON* and for an agent not using it. For each assistance session the graph plots the average precision value. We used the following numeric parameters to perform

[1] http://www.cs.waikato.ac.nz/~ml/weka

the tests: Minconf=0.8, Minsup=0.2, δ=0.5, number of association rules generated=2000, number of assistance actions=14.

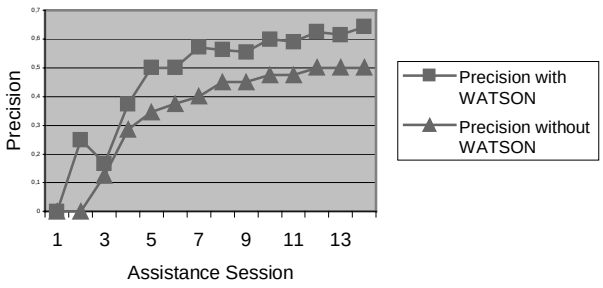

Figure 1. Evaluation of precision

4. Related Work

Related works have considered different approaches to decide among several possible agent actions. Some of them adopt a decision theory approach, such as [7] and [4], and others use confidence values associated to actions to decide what to do, such as [9] and [6]. However, although different factors are considered to compute the benefits and costs of different actions, and to calculate confidence values, the ones taking into account how the user wants to interact with the agent are missing in these approaches.

5. Conclusions

The *WATSON* algorithm enables interface agents to determine how to best assist users, personalizing in this way the interaction with each of them. The results obtained so far are quite promising, since our agents improve their assistance capabilities and their assistance actions tend to the users' needs.

References

1. R. Agrawal and R. Srikant - Fast Algorithms for Mining Association Rules – In Proceedings 20th Int. Conf. Very Large Data Bases (VLDB) – (1994) 487 – 499
2. R. J. Bayardo Jr. and R. Agrawal - Mining the Most Interesting Rules - In Proceedings of the 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – (1999)
3. Brown, S. and Santos, E. - Using explicit requirements and metrics for interface agent user model correction. In Proceedings of the 2nd International Conference on Autonomous Agents – (1998)
4. Fleming, M. and Cohen, R. – A utility-based theory of initiative in mixed-initiative systems – In IJCAI 01 Workshop on Delegation, Autonomy and Control: Interacting with Autonomous Agents – (2001) 58 - 65
5. Klementinen, M., Mannila, H., Ronkainen, P., Toivonen, H., and Verkamo, A. I. - Finding interesting rules from large sets of discovered association rules. In 3º Int. Conf. on Information and Knowledge Management – (1994)
6. Kozierok, R. and Maes, P. - A learning interface agent for scheduling meetings. In ACM-SIGCHI International Workshop on Intelligent User Interfaces – (1993) 81 - 93.
7. E. Horvitz - Principles of mixed-initiative user interfaces - In ACM SIGCHI - Conference on Human Factors in Computing Systems – (1999)
8. Liu, B., Hsu, W., and Ma, Y. - Pruning and summarizing the discovered associations - ACM SIGKDD International Conference on Knowledge Discovery and Data Mining – (1999)
9. Maes, P. – Agents that reduce work and information overload – Comm. of the ACM 37 (7) - (1994) 31 - 40
10. Srikant, R. and Agrawal, R. - Mining Generalized Association Rules - Future Generation Computer Systems, 13 (2-3) – (1997)