SADIO Electronic Journal of Informatics and Operations Research

http://www.dc.uba.ar/sadio/ejs

vol. 6, no. 1, pp. 12-20 (2004)

Using Association Rules to Learn Users' Assistance Requirements

Silvia Schiaffino [1,2] Analía Amandi [1]

[1] ISISTAN
Facultad de Ciencias Exactas – Univ. Nac. del Centro de la Pcia. Bs. As.
Campus Universitario – Paraje Arroyo Seco
Tandil, 7000, Bs. As., Argentina
{sschia,amandi}@exa.unicen.edu.ar

[2] Also CONICET

Abstract

Interface agents are computer programs that learn users' preferences to provide them personalized assistance with their computer-based tasks. In order to personalize the interaction with users, interface agents must 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 discover when the user needs a suggestion to solve a problem, when he requires only a warning about it, when he wants the agent to execute an action 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. Our algorithm uses association rules (AR) to discover associations among problem situations and a user's assistance requirements in a given application domain.

Keywords: interface agents, user profiling, association rules

1 Introduction

Providing personalized assistance to users with their computer-based tasks has been, so far, the main objective of interface agents. Interface agents are computer programs that have the ability to learn a user's preferences, interests and priorities, and help him to deal with one or more computer applications. A commonly used metaphor for understanding interface agent paradigm is comparing them to a human secretary or personal assistant who collaborates with the user in the same work environment [10].

To fulfill their main goal, the many interface agents that have been developed have focused their attention on learning a user's preferences, interests and priorities 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.

As it has been pointed out in [8], there are 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, poor timing of agent actions and inadequate attention to opportunities that allow a user to guide the invocation of agent services and to refine potentially sub-optimal results provided by the agent.

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 a situation of interest for the user, it has to decide among various possible assistance actions: warning the user about the problem and let him decide how to deal with it; suggesting him how to solve the problem; solving it on the user's behalf; or doing nothing. This decision will be mainly influenced by the knowledge the agent has to make a suggestion or to execute an action on the user's behalf, 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 discover when the user wants a suggestion to solve a problem, when he requires only a warning about it, when he wants the agent to solve the problem and when he wants the agent to do just nothing.

Consider, for example, an interface agent that assists a user with the organization of his agenda. Suppose that the user is scheduling a new event that overlaps with another event already scheduled. The agent can warn the user about this situation, it can also suggest the user how to solve it, i.e. suggest which event(s) the user should reschedule, or it can ignore the problem. However, the user does not always handle this kind of situations in the same way: he wants the agent's assistance in some contexts (e.g. an appointment with the doctor overlaps with an appointment with the dentist), but he wants to handle some other overlapping events by himself (e.g. two business meetings). 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 present a learning algorithm named *WATSON* that tackles the problem presented before. Our algorithm is based on the observation of a user's actions, particularly on a user's reactions to the agent's actions. Our algorithm uses association rules to determine what type of assistance the user wants in each problem situation that may arise. Association rules are used to model associations among problem situations or situations of interest and a user's assistance requirements. *WATSON* enables an interface agent to adapt its behavior and its interaction with a user to the user's assistance requirements.

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

2 *WATSON* 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. When an interface agent has to decide among various assistance actions to deal with a situation, the *WATSON* algorithm enables it to choose the one that is the most acceptable to the user in that particular situation. The following subsections describe in detail our algorithm.

2.1 Algorithm 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 executes to deal with the problem, the user feedback *UF* obtained after assisting the user and, when available, 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 underlying problem situation. 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 the user feedback the assistance experience can, sometimes, be evaluated as a success or as a failure. However, if the agent does not have enough information (e.g. no user feedback) the evaluation may be not available.

The output of our algorithm is a set of facts that indicate the assistance action the user requires in each 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 facts from the set of user-agent interaction experiences.

2.2 WATSON Overview

The *WATSON* algorithm uses the information contained in user-agent interaction experiences to formulate 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 a situation *P* occurs, the user will require an 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. The certainty degree of a hypothesis is computed using metrics from association rule mining, as we will see later.

Our algorithm uses 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, an interface agent can use them to discover the relationships among problem situations and the assistance actions a user requires to deal with 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 ← ∅
2. H ← ∅
3. Generate a set of association rules AR (using Apriori algorithm
for example) from Ex
4. AR_1 ← Filter out uninteresting rules from AR
5. AR_2 ← Eliminate redundant and insignificant rules from AR_1
6. H ← Transform rules in AR_2 into hypotheses (just notation
transformation)
7. FOR (i=1 to size of H){
        8. Find evidence for (E+) and against (E-) H
        9. Cer(H_i) ← compute certainty degree of H_i considering E+
        and E-
        10. IF (Cer(H_i) ≥ δ)
                11. F ← F ∪ H_i
   }
```

Fig. 1. *WATSON* Overview

The association rules generated from the set of user-agent interaction experiences are post-processed in order to derive useful hypotheses from them. 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. Once a hypothesis is formulated, the algorithm looks for positive evidence supporting the hypothesis and negative evidence rejecting it in order to validate it. The certainty degree of the hypothesis is computed taking into account both the positive and the negative evidence. Finally, facts are generated from the set of highly supported hypotheses. Figure 1 shows the main steps of our algorithm. The following sections explain in detail how we perform each of them.

2.3 Mining Association Rules from User-Agent Interaction Experiences

An association rule is a rule that implies certain association relationship 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 $I=\{i1,...,in\}$ 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 $X \cap Y = \emptyset$. X is the antecedent of the rule and Y is the consequent. The support of a rule $X \rightarrow Y$ is the probability of 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, 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[1].

2.4 Filtering Out Uninteresting and Redundant 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* $\rightarrow$ *user feedback, evaluation*". Other combinations of items are irrelevant since we are trying to discover which "problem situation-assistance actions" pairs have received a positive user feedback and were evaluated, in consequence, as a success.

We can use an intuitive approach [6] 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.

We (or a domain expert) can first classify attributes into a class hierarchy or taxonomy, since we might want rules containing attributes of a given class. For instance, in our example domain of calendar management, the events can be divided in different types: meeting, dinner, party, classes or courses, appointments with doctors and others. In turn, we can have different types of meetings, such as business meetings or school meetings, for example. Thus, we have the following generalizations:

- Business, School $\subset$ Meeting $\subset$ Event Type
- Friends, Family $\subset$ Party $\subset$ Event Type

We can express these generalizations by using taxonomies as shown in Figure 2.

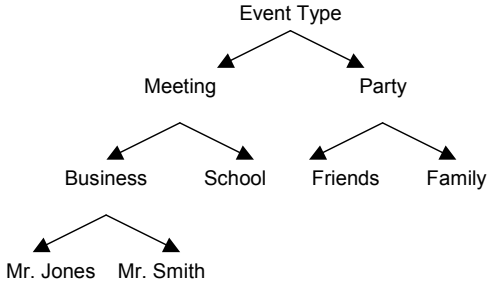

Fig. 2. Example of taxonomy

This taxonomy says that a meeting with Mr. Jones is a business meeting, a business meeting is a meeting, and a meeting is a type of event. When taxonomies are present, users are usually interested in generating rules that span different levels of the taxonomy. Users may want those rules containing descendants of a given item, for example. 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. All of them are classes, i.e. non-leaf nodes in the taxonomy.

Once we have filtered out those rules that are 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

[1] In this work we are not concerned about how association rules are generated. If readers want more information about association rule mining they can read [1], [11], [2].

the discovered rules [9]. 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, EventPriority=high, warning → failure, askforsuggestion [sup:40%,conf=91%]

If we know R1, then R2 is insignificant because it gives little extra information. Its slightly higher confidence is more likely due to chance than to true correlation. It thus should be pruned. R1 is more general and simple.

Besides, we have to analyze certain combinations of attributes in order to determine if two rules are telling us the same thing. For example, a rule containing the pair "suggestion, failure" and another containing the pair "warning, success" are redundant provided that they refer to the same problem situation and they have similar confidence values.

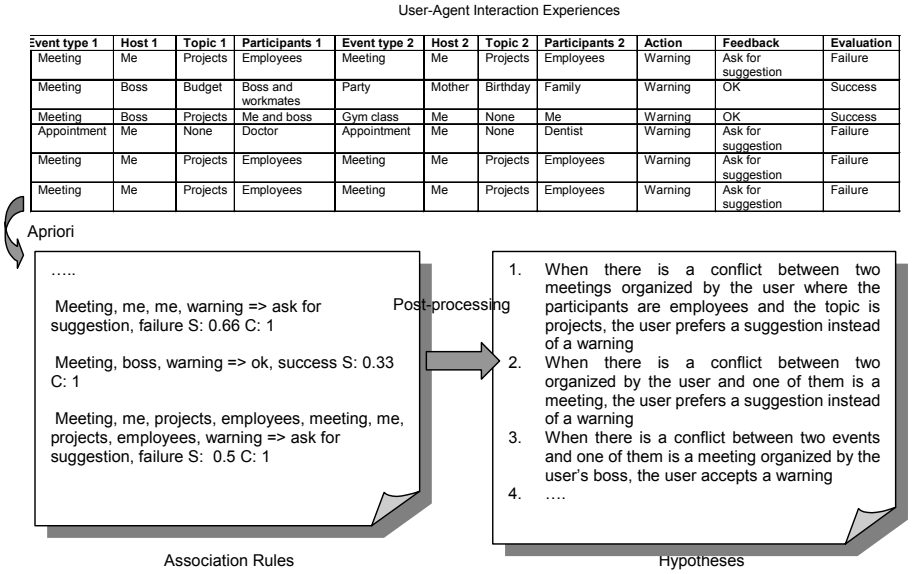

Fig. 3. Deriving hypotheses from user-agent interaction experiences

To avoid considering obsolete assistance experiences both to derive hypotheses and to validate them, our algorithm deletes old transactions from the database. The remaining rules are those that we consider to build hypotheses. Figure 3 shows, as an example, how a set of hypotheses is derived from a set of user-agent interaction experiences. In this example, the assistance actions involved are warnings about overlapping events.

2.5 Building Facts from Hypotheses

Once the *WATSON* algorithm has formulated a set of hypotheses it has to validate them. Our algorithm tries to validate a hypothesis by analyzing the evidence for and against it. If the hypothesis involves a warning, the evidence supporting it is composed of those assistance experiences in which the user has accepted the warning without asking for a suggestion, and those in which the user would have preferred a warning instead of a suggestion. The evidence against the hypothesis is given by those interaction experiences in which the user has requested a suggestion after the agent has warned him about a problem. If the hypothesis involves a suggestion, it is supported by those interaction experiences in which the user has requested the suggestion and those in which the user has accepted an autonomous suggestion from the agent. The evidence against this kind of hypothesis is given by those experiences in which the user has neglected suggestions.

The evidence for and against a hypothesis can be obtained by analyzing some of the association rules generated from the interaction database. For example, if we are trying to prove a hypothesis obtained from the rule "Situation X, warning → fbk1, success Sup:x Conf:y", and we have another rule that expresses "Situation X, warning → fbk2, failure Sup:z Conf:w" (where fbk1 and fbk2 are feedback items) we should discard the

hypothesis or at least adjust its certainty degree. Given a rule $X \rightarrow Y$ originating a hypothesis H, a rule of the form $X \rightarrow Z$ where $Z \cap Y = \varnothing$ is considered as a negative evidence, while a rule of the form $X \rightarrow Z$ where $Z \cap Y \neq \varnothing$ is considered as a positive evidence. The certainty degree of a hypothesis H is computed as a function of the supports of the rule originating the hypothesis and the rules considered as positive and negative evidence of H. The function we use to compute certainty degrees is shown in Equation 1, where α, β and γ are the weights of the terms in the equation (we use α=0.7, β=0.15 and γ=0.15), *Sup(AR)* is the support of the rule originating H, $Sup(E^+)$ is the support of the rules being positive evidence, $Sup(E^-)$ is the support of the rules being negative evidence, *Sup(E)* is the support value of an association rule taken as evidence (positive or negative), *r* is the amount of positive evidence and *t* is the amount of negative evidence.

$$Cer(H) = \alpha Sup(AR) + \beta \frac{\sum_{k=1}^{r} Sup(E+)}{\sum_{k=1}^{r+t} Sup(E)} - \gamma \frac{\sum_{k=1}^{t} Sup(E-)}{\sum_{k=1}^{r+t} Sup(E)} \quad \textbf{(1)}$$

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 2.

$$M_{precision} = \frac{number\ of\ correct\ actions}{number\ of\ actions} \quad \textbf{(2)}$$

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 the user expected in a given problem situation. The user's feedback (implicit or explicit) 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*. Half of the users used the agent with *WATSON* and the other half used the agent without our algorithm. We obtained the user feedback after each assistance action executed by each agent and we classified the assistance experience as a success or as a failure.

The graph in Figure 4 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 the tests: minconf = 0.8, minsup = 0.2, δ = 0.5, number of association rules generated = 2000, number of assistance sessions = 14, interactions between two consecutive assistance sessions = 5.

We can observe that the number of correct assistance actions is bigger for the agent using our learning algorithm. There is a pattern of improving performance in our agent's assistance capability, which indicates that the *WATSON* algorithm enables agents to improve their interactions with the users they are assisting.

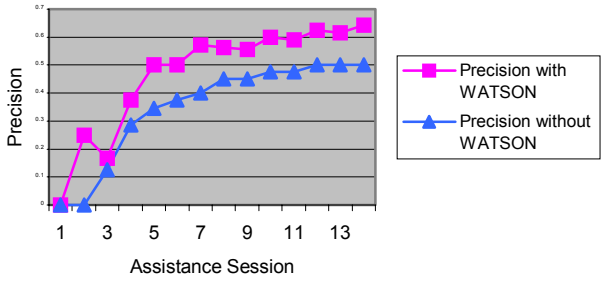

Fig. 4. Evolution of agents' precision with *WATSON*

4 Related Work

Related works, mainly in the area of mixed-initiative interaction, have considered different approaches to decide among several possible agent actions to assist a user. Some of them adopt a decision theoretic approach, such as [8] and [4], and others use confidence values associated to actions to decide what to do, such as [10] and [7]. However, although different factors are considered to compute the benefits and costs of different actions, and to calculate confidence values, those taking into account how the user wants to interact with the agent are missing in these approaches.

5 Conclusions and Future Work

Our work contributes to both the interface agent and the human-computer interaction areas. The WATSON algorithm enables interface agents to discover how to best assist users, personalizing and improving in this way the interaction with each of them. The results obtained so far are quite promising, since our agents have improved their assistance capabilities and their assistance actions tend to the users' needs.

In this work we have proposed the utilization of association rules to discover the relationships between a problem situation and the required assistance actions. We are now studying other techniques, such as Bayesian Networks and Influence Diagrams (an extension of Bayesian Networks) [5], in order to analyze which technique performs better regarding our algorithm.

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. E. Horvitz - Principles of mixed-initiative user interfaces - In ACM SIGCHI - Conference on Human Factors in Computing Systems – (1999)

6. Jensen, F. - Bayesian Graphical Models - Encyclopedia of Environmetics - (2000)

7. Klementinen, M., Mannila, H., Ronkainen, P., Toivonen, H., and Verkamo, A. I. - Finding interesting rules from large sets of discovered association rules. In Third International Conference on Information and Knowledge Management – (1994) 401 – 407

8. Kozierok, R. and Maes, P. - A learning interface agent for scheduling meetings. In ACM-SIGCHI International Workshop on Intelligent User Interfaces – (1993) 81 - 93.
9. Liu, B., Hsu, W., and Ma, Y. - Pruning and summarizing the discovered associations - ACM SIGKDD International Conference on Knowledge Discovery and Data Mining – (1999)

10. Maes, P. – Agents that reduce work and information overload – Communications of the ACM 37 (7) - (1994) 31 – 40

11. Srikant, R. and Agrawal, R. - Mining Generalized Association Rules - Future Generation Computer Systems, 13 (2-3) – (1997)