USER MODELLING FOR DIARY MANAGEMENT BASED ON INDUCTIVE LOGIC PROGRAMMING

Behrad Assadian, Heather Maclaren
Intelligant systems Lab, British Telecommunications, Adastral Park, Ipswich, United Kingdom

Keywords: Inductive Logic Programming,Personal Assistant,User Modelling,Software Agents,User Preferences

Abstract: Software agents are being produced in many different forms to carry out different tasks, with personal assistants designed to reduce the amount of effort it takes for the user to go about their daily tasks. Most personal assistants work with user preferences when working out what actions to perform on behalf of their user. This paper describes a novel approach for modelling user behaviour in the application area of Diary Management with the use of Inductive Logic Programming.

1 INTRODUCTION

There are a few applications of Inductive Logic Programming (ILP) (Muggleton 1994) to user modelling (Rogers 2000) already in existence. ILP is useful for user modelling as it allows the use of intensional background knowledge, hence we can incorporate `rules of thumb', and it maintains the transparency of the agent's actions to the user. i.e. they can look at the agent's model and understand what inferences it has made about them, thus maintaining their trust in their agent.

Current methods of ILP for user modelling deal with simpler concepts such as correcting a user's use of a unix shell (Jacobs 2000) or predicting which switches they would use for certain commands.

2 USER MODELLING

This paper proposes a system that learns sequence of activities from users diaries in order to model how such sequences should be arranged for future tasks.

A novel method of using ILP is proposed that splits the learning of the user model into several stages. It first produces results for each of these stages, then combines the results to produce a single user model.

The data is split into distinct clusters, each representing a sub-concept of the model to be learnt, and then the learning of each sub-concept is attempted separately. Each sub-concept is split into a number of separate learning problems which focus on a separate attribute within each data item and only require a subset of the available background information to solve, thus reducing the number of possible solutions that the ILP engine must consider to a size that it is capable of managing. The results of each learning problem are then combined to produce a set of rules, each of which contain range-restrictions for every attribute within each data item. Each set is then added into a database to produce the overall user model.

Meanwhile the clusters of data are also used to produce a series of probability distributions, which are stored for later use when querying the model.

The user modelling system consists of two parts: a construction engine which produces the user model, and a query engine which allows the user model produced to be used for the prediction of long sequences of tasks.

3 BUILDING USER MODEL

The initial user model worked on within the diary assistant (iMeeting) (Assadian, 2004) was the learning of sequences of pairs of tasks between tasks, e.g. if the user schedules a presentation on a particular project and they usually schedule some preparation time in before that presentation then we can learn this habit and either carry out the scheduling of preparation time automatically or make suggestions when the user enters the presentation task into the diary.

Let us suppose that we have data, which gives details of three sequences that the user is known to demonstrate: -

1. Putting in preparation time before an administration meeting.
2. Putting in preparation time before a project meeting or presentation.
3. Putting in travel time before paying a visit to another company.

Examples of the sequences take the form of a pair of tasks joined via the 'sequence' relationship.

All of the sequences(plus any others within the data set) are collected as a single set of examples, which must be split into separate clusters so that the learning of each sequence can take place separately.

We can perform the initial splitting of the data by using a bottom-up agglomerative clustering algorithm over the first task of each pair to produce a group of subsets, and then using the clustering algorithm again on each subset on the second task of each pair to produce the final clusters of examples, which will be used.

Each set is used as the example set for a series of different learning problems, each problem focuses on a different attribute within one or other of the tasks and attempts to find any specialisations which may be regarded as helping to characterize the particular sequence under examination. For example, the first learning problem would focus on the type of the first task of the pair in each example and would attempt to find any regularities amongst all the examples of the set for that particular attribute. Subsequent learning problems would focus on the subtype, sub-subtype, and duration of the first task individually, and then a further set of learning problems would focus on the individual attributes of the second task in the same manner.

Each learning problem requires positive examples, negative examples and background knowledge. The positive examples are the examples contained within the set that is currently under examination. The background knowledge used for each learning task is a subset of the entire set of background knowledge available, only those items of knowledge which directly refer to the attribute under examination are presented to the learner for each problem. It is this splitting of the available background knowledge into subsets in conjunction with the splitting of the overall learning problem into separate smaller problems (i.e. where the length of the clauses required is much smaller) which enables the learner to be able to tackle the overall problem of learning a user model as it reduces the number of possible hypotheses to be considered to a level which is manageable by the ILP engine.

A set of automatically generated 'negative examples' is produced for the attribute currently under examination. These are examples of pairs of tasks that the user would never produce and hence should not be thought of as being dependent on each other. Each set of negative examples only differs from the original data supplied by the user by a small amount, and all of the negative examples within a set differ from the original data in such a way that the ILP engine can use part of the provided background knowledge to successfully exclude all the negative examples from the solution that it produces.

If we were to generate a set of negative examples for the 'type' attribute then we would take a user-generated (positive) example:-

sequence:<travel/london,2hrs,10-00, thursday>, <visit/ericsson, 2hrs, 13-00, thursday>

And alter one of the values, producing:-

sequence:<admin/london,2hrs,10-00, thursday>, <visit/ericsson, 2hrs, 13-00, thursday>

This is repeated several times, using all of the examples within the set to generate negative examples. Values to be substituted into the attribute to be altered must satisfy the criterion that they must place the new example far enough away from the original example (using the distance measure used to produce the original clusters) that it could not be considered as part of the cluster of original examples. As it is likely that the amount of data with which we will be working will not be very large, the concepts being learnt may not be accurately characterised by the examples collected. This criterion allows a little more 'space' between the positive and negative examples and hence allows the learner to produce a rule which does not adhere so tightly to the exact details of the examples collected, hence a more general overall theory is produced which should provide better results when asked for predictions.

Generating of values for substitution where the variables being examined contain real values (for example the duration of a task) presents a further complication. In order to ensure that the values returned for a task prediction are accurate, the range of values that the variable is capable of being instantiated to must be limited. Values which are unacceptable as predicted values can be used to generate negative examples, but there may be cases where individual examples within the same cluster have values for a particular attribute which would be unsuitable if used within other examples in the same cluster. The two sequences listed below illustrate this problem:-

sequence:<travel/cambridge, 2hrs, 9-00, friday>, <visit/nokia, 1hr, 11-00, friday>

sequence:<travel/bath, 4hrs, 15-00, monday>, <visit/goulds, 5hrs, 10-00, tuesday>

The durations of the second task in each sequence are so far away from each other that we cannot allow them to be covered by a single rule. Negative examples for the first sequence would include values such as 5hrs and 6hrs. Negative examples for the second sequence would include values such as 2hrs and 1hr.

This would mean that generated negative examples may contradict other positive examples within the original set. We still need to restrict the range of values that the attribute can take, so the solution is to monitor for contradictions during the negative example generation process and if a contradiction occurs, split the set into a pair of subsets with the contradicted positive in one set and the positive from which the contradicting negative example was generated in the other. The other positive examples and their corresponding negatives are allocated to the new subsets according to whichever example they are closest to in terms of the attribute being examined. Negative example generation then continues, with further contradictions within the subsets resulting in further splitting actions, until all the positive examples have had negative examples generated from them. The sets are presented to the learner as separate learning problems and the results from each problem are added together to form a single set of possible specialisations for that attribute.

Once all the sub problems listed earlier have been formulated with the appropriate background knowledge and generated negative examples, and presented to the learner. We then have a collection of results for each attribute that must be combined to form the overall theory, which will characterise this particular sequence.

Some rules may contradict each other. This contradiction will be dealt with when we construct the rules, which form the theory. To construct the rules, we take the first two sets of results and combine them by adding all the rules from the first set to all of the rules from the second set.

The resultant set of rules is then combined with the next set of learning results in the same way, and the process is repeated for each set of results collected.

Each rule is tested for contradictions by evaluating it over the set of positive examples that it is supposed to characterise. If the rule does not cover any of the examples (i.e. it does not give the answer ‘true’ when given any of the pairs of tasks), then it is discarded. This test would remove rules containing contradictions. The set of rules is filtered to remove those rules subsumed by other rules, and each rule is then filtered to remove any redundant elements. Having performed these final filtering stages we are then left with a set of rules that form a theory that characterises the sequence we were trying to learn. This process is repeated for every cluster of examples that was initially generated and all the rules added to a collection which encapsulates the entire user model.

4 PREDICTING SEQUENCES

Having constructed our user model we then need to provide a means with which to use it..

When asked to suggest possible tasks, the query engine takes the task given and feeds it into the database of rules. It collects two lists; one of possible tasks to schedule before the user’s task, and one of possible tasks to schedule after the user’s task. Each list is then processed to find the most likely candidate for scheduling and the two answers returned. If there is no possible suggestion for either answer then an empty task that describes itself as ‘No Answer’ is returned as an indicator of this situation.

The generation of the Dirichlet distributions for use when rating answers makes use of information saved at the model learning stage. When the examples were originally clustered, a separate set of data was saved in which was stored the results of clustering the examples over the first task (task A) in the sequence and the results of clustering only over the second task (task B) in the sequence. This information represents the basis from which the set of distributions representing p(B|A) and p(A|B) can be calculated. As the method for generating distributions which deal with prediction of following tasks and distributions which deal with prediction of preceding tasks is the same, it will only be described from the point of view of predicting a following task.

All the possible stereotypes (Rich 1989) for task B can be determined by looking at the data representing the results of clustering over task B and taking the mode of each task B within a cluster as this will produce examples of the possible values for task B encountered so far. The data representing the results of clustering over task A will contain a set of examples for each distinct task A encountered. Each set can be used to create a Dirichlet distribution p(B|A) by counting the number of occurrences of each type of task B that follows the given task for that distribution and then normalising the counts to produce a probability. This version of the Dirichlet distribution uses a normal prior during construction, but leaves the possibility open to use of more biased priors later if required.

The most likely candidates from the two lists of tasks produced earlier are generated by sorting each list into sub lists of similar tasks (we may have generated several possible tasks which only differ by a very small amount, for example one task may have a preferred time half an hour later than another task), and then ascertaining the most suitable candidate from each sub list using the probability distributions created from the original set of examples collected. Two sets of distributions are created; one which describes P(B|A) and the other describes P(A|B). In both cases A is chronologically the first task in the sequence and B the second. If we are looking at the list of possible tasks which could follow that specified by the user then we would use the set of P(B|A) distributions as task A is given and we wish to ascertain the probability of each possible task B that has been generated. Conversely, if we are looking for a task which would precede the user's task then we would use the set of P(A|B). We are working with a set of distributions rather than simply one because we need to construct a separate distribution for each possible task given by the user (i.e. each distinct 'A'). Once we know the user's task then ideally we would concentrate on an individual distribution, however the distributions are created using stereotypes for different task types (the set of stereotypes used contains the mode of each cluster generated during the learning process) and the user's task may not match exactly any of the tasks over which the distributions are created. Therefore we pick all the distributions for which the distance from the base task to the user's task is closer than the threshold distance used at the clustering stage of the learning process.

Task ratings are generated by adding together the rating from each selected distribution in turn. For each distribution, the probability given to the stereotype that is closest to the task being rated is divided by its distance from the task to form the rating for that task. This allows us to attempt to distinguish between tasks that only differ by small amounts and is based on the idea of the influence of each point in the instance space represented by a stereotype degrading with distance (hence the sum of ratings, which is a simple method of acknowledging influence from more than one point). The tasks with the highest rating within each of the sub lists generated earlier are returned to be presented to the user as they are all valid sequences for the task originally entered.

5 CONCLUSION

The system described within this paper has demonstrated a new method of ILP application that enables this machine learning method to be used for user modelling within an Intelligent Diary (iMeeting). This augmentation allows iMeeting to make suggestions to the user based on previous observations.

Sparse data will always be a problem for any technique that attempts to make predictions based on previous experience, whether from a generalised version of the gathered information or from the examples themselves. In the case of ILP, small numbers of examples mean that the sort of generalised rules that were envisaged are not necessarily the ones that were constructed. Rather than creating a rule that covers 'all projects', a rule or group of rules which cover specific projects has been created. The system generates the most accurate hypothesis it can with the data presented, without continuous human intervention, and this may mean that the optimum set of rules is not always created. However, with more data, the quality of the hypothesis has been shown to improve.

REFERENCES

Muggleton, S, 1994. 'Inductive Logic Programming: Theory and Methods'. Journal of Logic Programming, Vol 19/20, pp629—679.

Rogers, S, 2000. 'The Learning Shell' Adaptive User Interfaces conference 2000

Jacobs, N, 2000. From Shell Logs to Shell Scripts. *The American Association for Artificial Intelligence 2000.*

Rich, E, 1989. Stereotypes and user modelling User models in dialog systems

Behrad Assadian 2003, An Automatic Meeting Scheduling for Mobile Users. In *ICEIS'2004, 6th International Conference on Enterprise Information Systems*