Experience With a Learning Personal Assistant

Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott[1], David Zabowski

School of Computer Science
Carnegie Mellon University

Abstract[2]

Personal software assistants that help users with tasks like finding information, scheduling calendars, or managing work-flow will require significant customization to each individual user. For example, an assistant that helps schedule a particular user's calendar will have to know that user's scheduling preferences. This paper explores the potential of machine learning methods to automatically create and maintain such customized knowledge for personal software assistants. We describe the design of one particular learning assistant: a calendar manager, called CAP (Calendar APprentice), that learns user scheduling preferences from experience. Results are summarized from approximately five user-years of experience, during which CAP has learned an evolving set of several thousand rules that characterize the scheduling preferences of its users. Based on this experience, we suggest that machine learning methods may play an important role in future personal software assistants.

1. Introduction

With the spread of networked personal computers, personal software assistants have rapidly appeared for a variety of tasks such as work management, information and mail organization, and calendar scheduling [12] [18] [20] [10] [21] [2] [4]. One can easily imagine future knowledge-based assistants that operate across networks as a kind of software secretary, providing services for work and home, such as paying bills, making travel arrangements, submitting purchase orders, locating information in electronic libraries, etc. In most such applications, user acceptance will depend on customizing the personal assistant to the particular habits and interests of the user. Just as with human secretaries, success will depend as much on knowledge about the particular user's habits and goals as on the specific set of operations the agent can perform. For example, an agent that sorts incoming mail will succeed or fail based on whether it correctly models the user's criteria for, say, the "Urgent" mail category.

How can we build software assistants that can be easily customized to individual users? Many programs provide simple parameters that allow users to customize behavior explicitly. For example, text editors allow users to set default type fonts and directories, while email sorting programs allow users to declare keywords that indicate which messages should be sorted to which categories. These approaches are limited, however. As we scale up to more sophisticated assistants, the ability of most users to program their preferences explicitly will decrease. Customizing an email sorter to accommodate one's personal notion of an "Urgent" message, for example, requires a detailed articulation of a fairly subtle concept. Furthermore, even if users

---

[1]John McDermott is affiliated with Digital Equipment Corporation.

[2]A revised version of this paper will appear in the Communications of the ACM, July 1994.

are willing and able to put in the effort to initially customize their assistants, they may be unwilling to continually update this knowledge. A message about a particular business contract might be quite urgent before an approaching contract deadline, but not necessarily urgent after this deadline.

The thesis of our research is that software assistants can automatically customize to individual users, by learning through experience. We are currently exploring this thesis by developing learning software assistants for calendar management, electronic newsgroup filtering, and email negotiation. In each of these applications, our approach can be summarized as follows:

- Provide a convenient interface that allows the user to perform the task (e.g., an editing and email interface to an online calendar).
- As the system is used, treat each user interaction as a training example of this user's habits (e.g., each meeting scheduled by the user reflects their preference for the duration, time, location, etc., of this type of meeting).
- Learn general regularities from this training data, and use this learned knowledge to increase the services offered by the software assistant (e.g., provide interactive advice to the user as they schedule future meetings, or offer to negotiate specific meetings on behalf of the user).

We refer to this kind of interactive learning assistant as a *Learning Apprentice* [11]; that is, an interactive assistant that acquires knowledge through routine use by observing users' actions. Negroponte [13] and Kay [8] were among the first to recognize the potential value of personal learning assistants. An early example of a learning apprentice is the LEAP system [11] which dealt with the domain of VLSI digital logic design. Earlier descriptions of the calendar apprentice described in the current paper can be found in [7] [4], and a related effort to develop a learning calendar assistant is described in [10]. A learning apprentice designed to make browsing more efficient through online information sources such as library catalogs is under development by Holte and Drummond [6]. Nakauchi and Anzai have developed a prototype system that learns to help users fill out purchase orders by learning preferences such as which vendor to use for which parts [12].

Semi-automated methods for user customization are also being studied. For example, Bocionek and Sassin propose a dialog-based learning method [2] to allow users to train interface agents by demonstration. Kodratoff and Teucci investigate an iterative approach to learning apprentices in their work on the DISCIPLE System [9]. DISCIPLE has been used for applications ranging from designing manufacturing technologies to aiding the manager of a computer center. Sheth and Maes have explored the use of simulated evolution to evolve agents for personal information filtering [21]. Exemplar-based learning for apprentice systems has been studied by Bareiss et al [1] in the Protos system which they apply to the domain of clinical audiology. And a number of researchers explore the use of machine learning for information retrieval [19], filtering [5] and cataloging [20].

The next sections of this paper present a case study of a learning apprentice for calendar management called CAP. CAP provides an editing and email interface to an online calendar. It learns users' scheduling preferences through routine use, enabling it to give customized

scheduling advice to each user. Several copies of CAP have accumulated approximately five user-years of experience from a handful of users. Below we describe CAP, its learning methods, and experimental results from the fielded system. The final section describes lessons learned from this case study regarding the potential of machine learning methods for customizing software assistants in other task domains.

2. The Calendar Apprentice

CAP is a learning apprentice to assist the user in managing a meeting calendar. It provides interactive access to an online calendar and to electronic mail. Users can edit the calendar by adding, deleting, moving, copying and annotating meetings, and can mark various calendar events as either tentative or confirmed. Other user commands instruct CAP to send electronic mail meeting invitations (e.g., for meetings marked tentative) or meeting reminders (e.g., for confirmed meetings) to the attendees of the specified meeting. User commands are also available for tagging days with text notes, for altering the calendar screen display and for printing the calendar in various formats.

What kinds of assistance might such a system provide to its user? Consider the analogy to a human secretary who might assist someone in managing a calendar. A human secretary, when they first begin to work with a particular person/user, might carry out similar detailed instructions to add or alter individual meetings, or to send email reminders. Over time, however, the work burden will shift as the human secretary learns enough about the user's scheduling preferences to take over routine aspects of the task. For example, the secretary will learn when, where, and for how long the user typically meets, depending on the type of meeting, the attendees, etc. The secretary will learn which meetings may be moved or cancelled in order to make room for higher priority meetings. This acquired knowledge will alter the interaction between the secretary and user, so that the user can give more brief instructions such as "please set up a meeting with Joe for next week", trusting that the secretary has the appropriate knowledge to fill in the detailed location, time, and duration of the meeting. Furthermore, this acquired knowledge of user scheduling preferences will allow the secretary to negotiate with others to arrange meetings on the user's behalf, effectively offloading work from the user.

Our goal for CAP is to have it exhibit a similar learning behavior, where each copy of the system should learn the scheduling preferences of its user, and evolve gradually from a passive editing interface to a knowledge-based assistant capable of interacting more intelligently with the user and offloading the work of meeting negotiation from the user. To date, our focus has been on developing the learning methods, and on using learned knowledge to provide interactive advice as the user adds new meetings to the calendar. CAP currently learns rules that enable it to suggest the meeting Duration, Location, Time, and Date. More recently we have begun developing a system to make use of this learned knowledge to negotiate meetings semi-autonomously via email on behalf of the user.

2.1. System Organization

Figure 2-1 depicts the interface to CAP seen by the user[3]. In this figure, the user is viewing a

[3]This interface is built on Gnu-Emacs, and communicates with an underlying Lisp process which provides CAP's learning and inference capabilities.

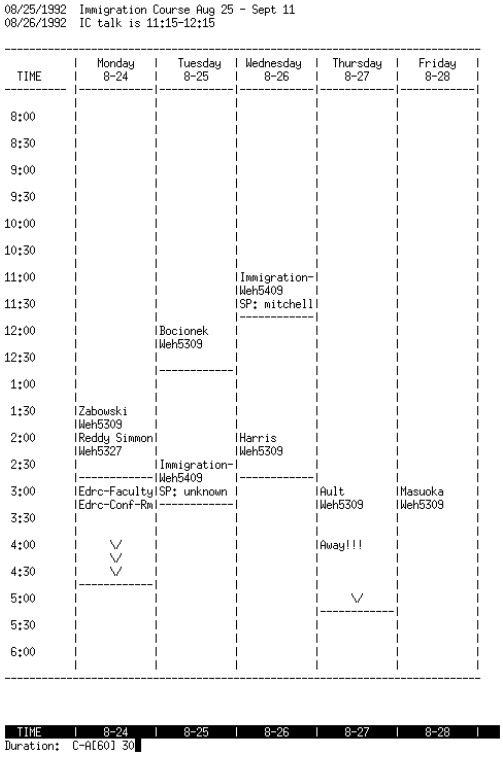


Figure 2-1:

The CAP User Interface. The user is adding a meeting while displaying a particular week of the calendar. At the bottom of the screen CAP prompts the user for the meeting Duration, offering advice (60 minutes) based on its learned scheduling preferences. In this case, the user overrides the advice.

particular week of the calendar, and is in the process of adding a new meeting. In general, when the user adds a new meeting, CAP prompts in sequence for the meeting Type, Attendees, Date, Duration, Time, Location, and for whether the meeting is Tentative or Confirmed. At the bottom of the screen in the figure, CAP is prompting the user for the Duration of the meeting and offering the suggestion that this meeting be allocated 60 minutes. This suggestion is derived from a previously learned rule that matches the known features of this new meeting (i.e., those features for which the user has already been prompted, plus any features inferred from these). The user may accept this advice or override it by entering the desired value. In this figure, the user is overriding the advice, and instructing the system to allocate 30 minutes for this meeting.

Whenever the user accepts or overrides CAP's advice, a training example is captured that is used for subsequent learning. The meeting described in Figure 2-1, for instance, provides a training example of the general concept, "meetings for which Duration is 30 minutes." A representative training example meeting acquired by CAP from such an interaction is shown in

request-5-27-1992-48:
attendees: thrun
event-type: meeting
date: (29 5 1992)
time: 1430
duration: 30
location: weh5309
confirmed?: yes

displayed-week: (25 5 1992)
action-time: 2915977709
action-date: (27 5 1992)
previous-request: request-5-27-1992-13
previous-prompt: confirmed=yes

position-attendees: project-scientist
previous-attendees-meeting: request-5-20-1992-1
next-attendees-meeting: none
lunchtime?: no
number-of-attendees: 1
cmu-attendees?: yes
attendees-in-toms-group?: yes
known-attendees?: yes
day-in-week: friday
end-time: 1500
busyness-of-attendees: 2
single-attendee?: yes

Table 2-1:

Typical Training Example of a Calendar Meeting. The top set of features corresponds to attributes of the meeting entered by the user (or accepted, in response to the agent's advice). The middle set includes automatically collected attributes characterizing the current state of the computational environment. The bottom set includes additional features inferred by the system, from background knowledge about people and about the semantics of the basic calendar attributes.

Table 2-1. Notice that the training example includes attributes beyond those provided by the user (e.g., Date, Time); it also includes attributes of the current calendar state (e.g., which week is displayed to the user), as well as attributes inferred from background knowledge about the domain (e.g., the position of the attendees, the previous meeting involving the same attendees).

From this kind of training data CAP learns rules that suggest the Duration, Location, Day-of-Week, and Time of meetings. A sample of these learned rules is shown in Table 2-2. For each rule, statistics are maintained that summarize its performance both on the training data that gave rise to the rule, and on subsequent examples. Rules of each type (e.g., those that predict Duration) are kept on a list sorted by their past performance, with advice generated by the

```
If   Position-of-attendees is Grad-Student, and
     Single-attendee? is Yes, and
     Sponsor-of-attendees is Mitchell;
Then Duration is 60.
[Training: 6/11  Test: 51/86]

If   Group-name is EDRC-Directors;
Then Duration is 90.
[Training: 6/6  Test: 31/38]


If   Position-of-attendees is Faculty, and
     Department-of-attendees is SCS, and
     Number-of-attendees is 2;
Then Location is Weh5220.
[Training: 2/3  Test: 13/16]

If   Position-of-attendees is Grad-student;
Then Location is Weh5309.
[Training: 21/21  Test: 56/59]


If   Seminar-type is THEO;
Then Day-of-week is Monday.
[Training: 6/6  Test: 8/8]


If   Department-of-attendees is EDRC, and
     Day-of-week is Friday;
Then Time is 8:30.
[Training: 5/5  Test: 18/19]

If   Course-name is 16-741;
Then Time is 9:30.
[Training: 35/35 Test: 59/60]
```

Table 2-2:

Typical Learned Rules. Each evening CAP typically learns 5-20 new rules for each meeting feature. Rules are prioritized in the system based on their performance over the training data and during subsequent use. For example, the first rule was correct in 6 of the 11 training examples from which it was formed, and in 51 of 86 subsequently encountered examples to which it applies.

topmost rule on the list that matches the new meeting.

CAP provides advice to the user regarding meeting Duration and Location based directly on these rules. CAP's advice regarding meeting Date (e.g., March 4) is inferred from rules that suggest Day-of-Week[4] (e.g., Thursday). For example, if today is date D, and the rules predict

[4]Day-of-Week is predicted rather than Date because rules that predict a particular date such as March-4-1994 are too specific to be of lasting value.

that the meeting should be on a Wednesday, then the date suggested by CAP is either the first Wednesday following D, or the Wednesday in the week currently displayed by the user, whichever is later. CAP's advice regarding the Time of a meeting is taken directly from its rules, unless the suggested time slot is already booked. In this case, CAP recommends the closest alternative time for which a calendar opening of the desired Duration is available.

In addition to its learned rules about user scheduling preferences CAP also uses factual knowledge about individual attendees of meetings. In particular, for each novel attendee that appears on the calendar it acquires that attendee's Institution, Position, Department, Email-Address, and Supervisor. At present, this information is requested from the user when the attendee is first encountered. Alternatively, this information could be obtained automatically from online databases containing personnel information. We are currently extending the system to make use of such databases available in our environment. As can be seen in Table 2-2, these attributes of attendees are used in the preconditions of the learned rules.

2.2. Learning Method

Each night, CAP automatically runs a learning process to refine the set of rules that it will use to provide advice on the following day. This process applies a learning procedure similar to ID3 [14, 15] (see Appendix) to learn a decision tree from the most recent training data. Each path through the decision tree is converted into a rule. In order to improve generality, rule preconditions are pruned when this improves rule performance. This learning procedure uses training examples such as the one shown in Table 2-1, to acquire rules such as those shown in Table 2-2. Each evening CAP learns approximately 40-50 rules for each of Duration, Location, Time, and Day-of-Week, of which 5-20 may be novel (i.e., different from previously learned rules). These learned rules are merged into the previous lists of rules, removing duplicates and sorting based on their measured accuracy. CAP's learning process is summarized in Table 2-3.

A number of design choices arise in defining the learning procedure for software assistants such as CAP. The first design choice is "Which learning method should be used?" We initially considered two inductive learning methods: decision tree induction [14] and artificial neural network (ANN) backpropagation [16]. While we found experimentally that these two learning methods produced comparable accuracy when trained on the same set of data [4], we came to prefer decision trees for two reasons. First, decision tree learning produces collections of rules that are intelligible to humans (see Table 2-2), whereas learned weights in ANN's are difficult to interpret. Human readability will be important in systems where humans may wish to inspect, edit, or approve learned knowledge. Second, the rules output from decision tree induction provide a *piecewise* representation of learned information, in contrast to the *monolithic* representation used in ANN's. This enables the system to monitor performance of individual rules over time so that outdated rules can be rejected while other highly useful rules are retained.

As second design choice is "How large a window of training examples should the learner utilize?" A large window of training data is desirable in order to assure that any regularities found by the learning procedure will be statistically significant. Too large a window of training data, however, will include very old data that may not be representative of current scheduling regularities. For example, in our university environment the onset of summer break often leads to major shifts in the scheduling preferences of users. We have found empirically that a window of 180 examples works well for our current learning method and users. For faculty users this

1. Update the performance statistics for each current rule, to include performance on all new training example meetings.
2. Window-Examples ← the most recent 180 training example meetings.
3. Training-Examples ← 120 examples selected at random from Window-Examples
4. Test-Examples ← Window-Examples - Training-Examples.
5. For each feature f in {Duration, Location, Day-of-Week, Time}
   - Learn a decision tree to predict values of feature f, using the ID3 algorithm (set Appendix) applied to Training-Examples
   - Convert each path of the learned decision tree into a rule
   - Remove any rule preconditions that do not result in decreased rule performance for this rule over either the Training-Examples or Test-Examples.
   - For each new rule, record the number of positive and negative examples it matches from Window-Examples
   - Sort each new rule into the previous rules for feature f, based on their accuracy as measured in the previous step.

Table 2-3: Learning Procedure in CAP. Each night CAP runs the above procedure to learn new rules and merge these with its current rule sets.

corresponds to roughly two calendar months, which is less than the length of a semester. Given the changing environment in which CAP must learn, methods that could learn reliably from shorter windows of data would be very useful.

A third design choice is "Which vocabulary of meeting attributes should be considered by the learner?" Hundreds of attributes, or features, are potentially available to CAP to describe each training example meeting. For example, available attributes include "the previous meeting with these same attendees", "the department of the attendees of the meeting before the previous meeting with these same attendees", etc. One would like the learner to consider all these attributes in its search for general rules. However, as one increases the number of attributes one must also increase the number of training examples in order to maintain a fixed level of learning performance (more data is required to select reliably among the larger set of candidate hypotheses). We have developed a method called Greedy Attribute Selection [3] that automatically selects which attributes to use for future learning by determining which attributes would have lead to most successful learning in the past. This technique can be used to dynamically customize the learning procedure to each user and each type of learned rule.

Yet another design choice in defining the learning procedure is "How should new learned rules be integrated with existing rules?" Given that nightly learning uses only a fixed window of recent data, and given that some previously learned rules characterize useful regularities that might not be apparent in recent data, one would like to combine new learned rules with previously learned rules that have been found to be successful in practice. CAP accomplishes

this by maintaining statistics on the number of correct and incorrect predictions of each rule over the training data and during subsequent use. Rules are sorted based on these performance statistics. The net effect is that old rules that have been found empirically to be of use rise to the top of the list, along with new rules that perform well on the training data. Old rules that are found ineffective in practice quickly drop down the list. A separate list is maintained for each type of rule (e.g., for rules predicting meeting Time).

3. Experimental Results

CAP has been used to varying degrees by half a dozen users. Two of these users, university faculty members who have fairly busy calendars, have used CAP as their sole calendar record for over 16 months. The experimental results presented here are taken from the data collected by CAP while in routine use by these two subjects.

How well do CAP's learned rules perform in practice? One way to answer this question is to plot the percentage of CAP's advice that coincides exactly with choices made by the user over the period of time that the system has been in use. This data is plotted in Figure 3-1, which shows the accuracy of advice for meeting Day-Of-Week, Duration, Time and Location, for both user A and user B[5]. In these plots, the solid line indicates the accuracy of advice based on CAP's learned rules. For comparison, the dotted line indicates the accuracy of advice based on a dynamically computed default value: the most common value of the feature over the most recent 180 examples. The accuracy plotted for a particular date is the measured accuracy of advice for the 60 meetings following that date. User A's graphs cover the period March 1992 through December 1993, whereas user B's cover November 1992 through November 1993.

What conclusions should be drawn from the results of Figure 3-1? One conclusion is that it is indeed possible for the system to learn rules that characterize scheduling preferences to some significant degree. The accuracy of learned advice varies significantly from feature to feature, and user to user, from an average accuracy of 69% for Duration, User B, to a low of 31% for Time, User B. Note that for an uncrowded calendar, it is particularly difficult for the agent to predict the user's choice of meeting Time, since there may be many equally acceptable times, from which the user might choose arbitrarily. The average accuracy of learned rules across all features and both users is 47%, compared to an average of 24% for advice based on default values. In these graphs, advice is considered accurate only if it is identical to the user's input.

Notice that the accuracy of CAP's advice varies over time, reflecting the dynamic nature of the domain and the need for updating user-specific scheduling preferences. For both faculty users, the periods of poorest performance correlate strongly with the semester boundaries in the academic year (i.e., January, June, September). For example, notice the drop in performance for both users during August-September, as the new semester begins and old scheduling priorities are replaced by new ones. CAP's performance typically recovers following these dips, as it learns new rules reflecting the user's new scheduling regularities.

---

[5]The learning procedure reported here is somewhat improved over the procedure in use while the training data was originally collected. Thus, the user did not necessarily see the same advice that our subsequent learning experiments would produce. Because the user decisions might have differed had they seen this advice, it is possible that the results of this experiment are overly pessimistic.

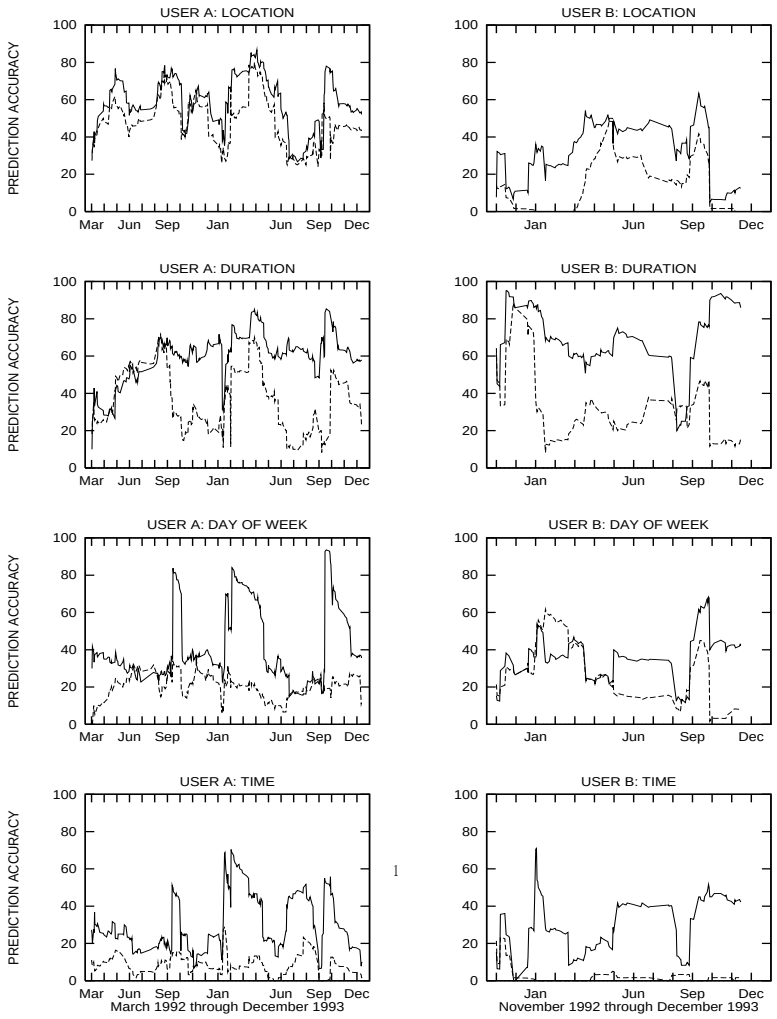


Figure 3-1:

Accuracy over time for the Day-Of-Week, Duration, Time, and Location prediction tasks. The graphs on the left are for User A, and those on the right for User B. Solid lines indicate the accuracy of advice generated by CAP's learned rules. Dotted lines indicate the accuracy obtained by suggesting the default value for this feature.

While it is clear that CAP learns something of use, it is disappointing that the accuracy of CAP's advice is not higher. While this level of accuracy is acceptable for providing interactive advice which the user can override while adding new meetings, our eventual goal is to have such assistants offload work from the user by autonomously performing portions of the user's workload. What can be done to increase the accuracy of CAP's knowledge to a level at which it might be entrusted to automatically schedule meetings? While we believe the learning method might be improved in various ways, we expect a more promising approach is to have the agent focus only on meetings for which it is confident in its predictions. We have found that there are certain meetings that can be predicted fairly accurately (e.g., meetings between our faculty subjects and their student advisees), whereas other meetings are not at all predictable from past experience (e.g., one-time off-campus trips). If CAP could successfully learn to discriminate those meetings which it can confidently handle from those which it cannot, then it might become a useful autonomous scheduler of *routine* meetings, and simply forward difficult cases to its user, just as a human secretary tends to operate autonomously in many cases and interacts with the user in other problematic cases.

Can CAP successfully distinguish cases in which it should trust its learned rules from those in which it should not? Figure 3-2 provides evidence that it can, by relying on observed past performance to attach confidences to individual rules. In this experiment, rules learned for Location for subject A on the evening of Sept 30, 1992 have been sorted based on their accuracy over the training data. The "1" in the figure shows the accuracy of the top-ranked rule in this sorted list, applied to the 60 subsequent meetings following training. Notice this single rule covers 33% of these subsequent meetings, with an accuracy of 95%. We use the term *coverage* to indicate the proportion of examples for which a prediction can be made. The "3" in the figure shows the coverage and accuracy of CAP if it uses the three highest ranked rules, and so on. Omitted numbers (e.g., 2) indicate that the corresponding rule set performed the same as the next shorter set (e.g., 1). As the size of the rule set is increased to include all 52 learned rules, the coverage grows from 33% to nearly 100%, while the average accuracy of advice drops from 95% to 67%. Notice that the accuracies plotted in the earlier Figure 3-1 correspond to the rightmost point in Figure 3-2. As this figure illustrates, CAP can increase the average accuracy of its predictions by offering advice only when its top-ranked rules apply, at the cost of reduced coverage.

How would CAP's performance from Figure 3-1 change if it offered advice only when the matching rule had a past accuracy of at least N%? Figure 3-3 shows the average accuracy of advice for the same eight plots, as N is varied from 0% to 95%. The rightmost points in each plot indicate the accuracy and coverage of all rules, averaged over the same time period as in Figure 3-1. Points further to the left indicate the increasing accuracy and decreasing coverage of CAP's advice, as the threshold N is increased from 0% to 95%. For example, for User A, the average accuracy of advice improves from 52% to 79%, while rule coverage drops from an average of 99% to 32%.

4. Conclusion and Prospects

CAP provides a case study for exploring the thesis that machine learning techniques can lead to self-customizing software assistants. The primary lessons of this case study include:

- It is feasible to automatically learn user-specific meeting preferences from passive

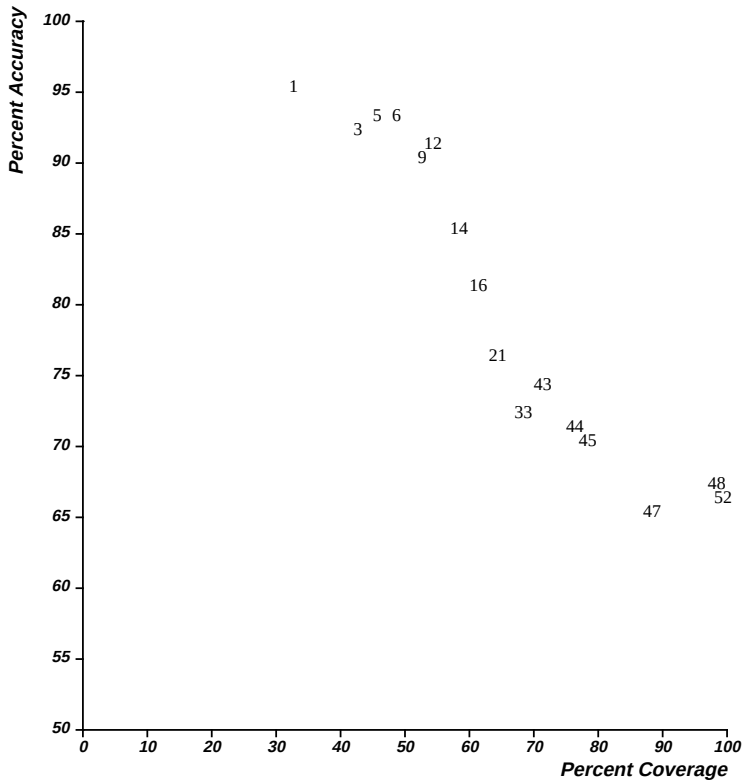


Figure 3-2:

Tradeoff of Coverage for Accuracy in Learned Rules. Prediction accuracy is plotted for different subsets of CAP's rules. Learned rules from Sept 30, 1991 are first sorted by their performance over the training data, then used to predict the 60 subsequent meetings. The "1" in the curve indicates the accuracy and coverage of the top ranked rule taken alone. The "3" indicates the accuracy and coverage of the top 3 ranked rules, and so on. This curve illustrates the potential for the learner to increase accuracy by limiting coverage.

observation to an accuracy that significantly surpasses simpler approaches such as computing default values.

- Learned rules in CAP are typically understandable to users, allowing for the possibility of users evaluating, augmenting or editing them.
- While rules learned by CAP are useful for providing interactive advice to be approved or overridden by the user, they are not sufficiently accurate to support autonomous negotiation of all meetings by the agent on the user's behalf.
- One method for improving the accuracy of CAP's advice is to allow it to volunteer advice only when the applicable rules are ones that have been relatively accurate in the past. This suggests a more likely role for a software agent than the complete automation of user workload: an agent might select and autonomously handle the subset of situations for which it has high confidence, referring difficult, non-routine cases to the user. We expect this kind of shared responsibility will be a more useful

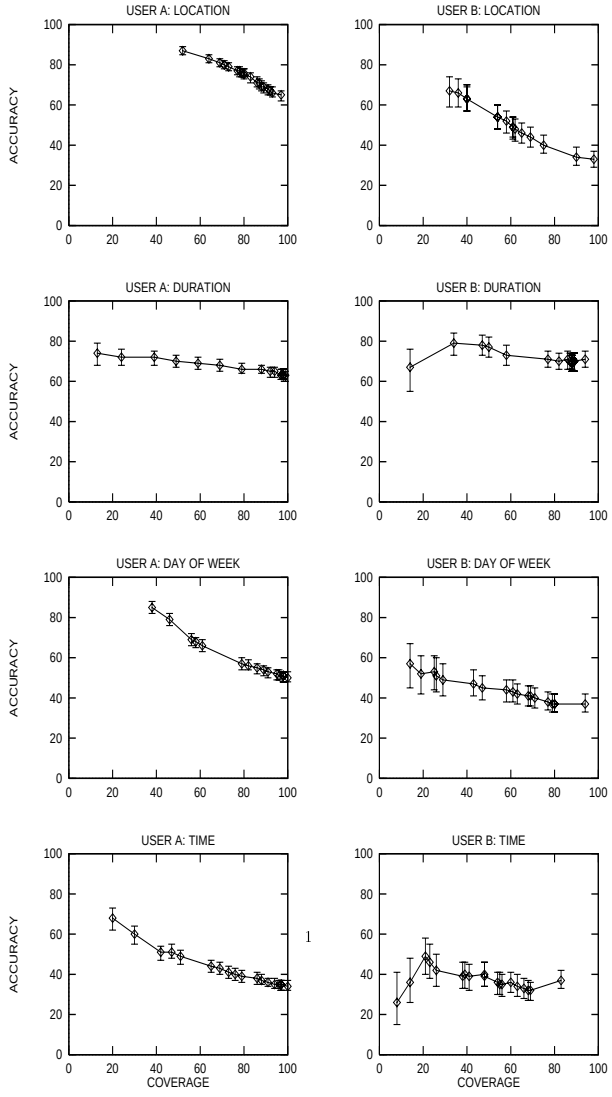


Figure 3-3:

Average Accuracy versus Coverage. This figure shows the accuracy versus coverage tradeoff for the eight prediction tasks (two users, four rule types) of Figure 3-1. Whereas Figure 3-2 showed this tradeoff for one day, this figure shows the *average* accuracy for this tradeoff over several years of use. For most of the users and prediction tasks, accuracy increases smoothly as coverage is reduced.

model for practical software assistants.

- One effective method for learning within a changing environment is to measure performance of individual rules over time, and to continually merge and sort new learned rules with old, based on their empirically determined accuracy. The modularity of rules (compared to neural networks) is important for this process.

What other tasks might be suited to the learning apprentice approach taken by CAP? Several characteristics of the calendar management task appear important to the approach: First, in order to collect initial training data the system must provide a service that makes it attractive to new users even before it has acquired any knowledge about this user. In the case of CAP, the system is sufficiently useful as a standard, unintelligent calendar editor. Many tasks that are routinely performed online without user customization satisfy this property (e.g., form-filling tasks such as entering purchase orders). Second, perfect advice and complete coverage by the assistant should not be a necessity. Third, the time interval needed to acquire sufficient data for reliable learning (the learning time constant) must be shorter than the time interval over which learned regularities remain stable (the task time constant). In the case of CAP, for our faculty users, the learning time constant is a month or two, and the task time constant approximately four months (one academic semester). Finally, the attributes on which user decisions are based must be sufficiently *observable* to the system. In the calendar task, decisions about meeting duration, location, etc., depend on attributes such as the state of the calendar and features of the meeting attendees (both observable to the system). Unfortunately, they also depend on features such as which meeting rooms are available -- features that are not currently observable to CAP, and that result in limits on its performance. Two other tasks that we believe satisfy the above properties, and that we are currently pursuing, include a newsgroup reader that learns which types of articles its users find interesting, and an email-based assistant that learns routine types of messages and conditions under which the user sends them.

Our experiments suggest a number of topics for future research. As noted above, one important goal is to extend CAP to the point that it can autonomously negotiate via email to arrange routine meetings while forwarding non-routine meeting requests to the user. In addition to the obvious issue of improving the accuracy of learned knowledge, this also raises research issues related to transfer of authority between user and system. For example, how can user supervision of the agent be best organized to allow the user to evaluate the agent's evolving decision making strategy, and to allow a gradual transfer of authority and release of supervisory control as both the user and assistant gain confidence in the assistant's evolving capabilities?

A second research issue involves reducing the time constant of learning. One approach might be to learn more abstract, but more temporally stable knowledge such as "Each seminar in a seminar series is typically held in the same Location, and has the same Duration." This type of learned knowledge (called determination knowledge [17]) could enable the agent to infer the general rule for Location and Duration for a new seminar series, as soon as the first seminar in the series was encountered. Another approach to reducing the learning time constant might be called cooperative learning, e.g., learning rules by pooling training data from multiple users. For example, by combining training data from multiple faculty users of CAP, one might learn rules that are valid to that class of users (e.g., IF the meeting is the Admissions-Committee-Meeting, THEN the Location is Room-A). By pooling training data from N users, the learning time constant for such rules could be reduced by a factor of N.

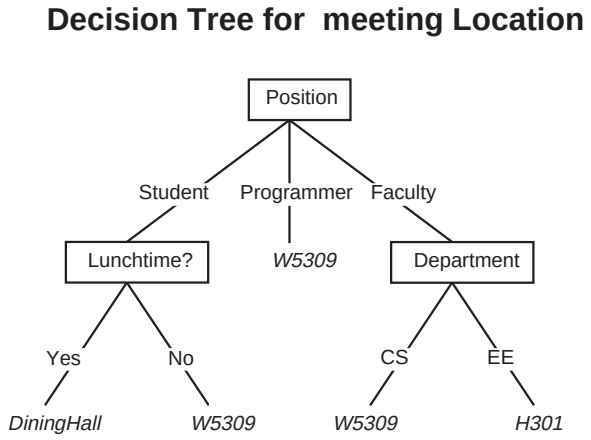
Increasing the observability of the world by interfacing to other online information sources, including other agents, is another useful research direction. Allowing the agent access to other users' calendars, to room reservation databases, or to personnel databases would increase the number of potentially relevant attributes that could be considered by the agent in its decision making. Of course, an increase in the number of such attributes will generally lengthen the time constant of the learner, raising the additional research issue of learning efficiently from large sets of potentially relevant attributes.

CAP provides a demonstration that machine learning methods can acquire many of the calendar scheduling preferences of individual users, and can also estimate the reliability of various learned rules. While these results are encouraging, we are just beginning to collect sufficient data to be able to understand the capabilities and difficulties in developing self-customizing systems; it remains to be demonstrated that knowledge learned by systems like CAP can be used to significantly reduce their users' workload. Our research plan is to extend CAP to negotiate meetings on its users' behalf, and to explore additional tasks including learning users' newsgroup reading preferences, and learning strategies for email-based work-flow assistance. Given the potential impact of a success in this area, we anticipate a flurry of experiments in machine learning approaches to self-customized assistants over the coming years.

5. Acknowledgments

We gratefully acknowledge the contributions of former members of this project, including Siegfried Bocionek, Lisa Dent, Jean Jourdan, Liam McDermott, and Owen McDermott. Ken Lang, Matt Mason, and Doug Reiken provided helpful comments on an earlier version of this paper. This research is sponsored in part by the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330, and by a grant from Digital Equipment Corporation.

I. Inductive Inference of Decision Trees



A *decision tree* organizes the problem of classifying an object, or *instance*, into a series of questions about the object. For example, the above decision tree classifies calendar meetings according to meeting Location. An instance is classified by starting at the root node of the decision tree and following branches, based on the results of individual tests, until a leaf is reached, at which point the instance is assigned to the class with which the leaf is labeled.

ID3 is an algorithm for learning decision trees from examples. The original algorithm is due to Quinlan [14], though many variants have since been developed. Given a set of training examples, ID3 produces a decision tree by growing it top-down, at each point greedily picking the test attribute that best classifies the training examples. This algorithm can be summarized as follows:

```
Grow-tree(training-sample)
  If Terminate-condition(training-sample)
    Return an appropriately labeled leaf node
  Else
    Let F = Choose-best-feature(training-sample)
    Let N be a decision node testing F
    For each value V of F in training-sample
      Let subsample = all instances I, such that F(I) = V
      Attach a branch of N to Grow-tree(subsample)
    Return N
```

One often adequate termination condition (the first conditional block) is to stop when all instances in a subsample have the same class.

As it grows the tree top-down, ID3 repeatedly chooses a feature for the next test node in the tree. It chooses a feature for testing (the second block) which most reduces ambiguity--increases *purity*--in the training sample. ID3 estimates purity by measuring the entropy of a sample. In the two-class case, given a training sample T with n instances, c_1 of which belong to Class 1 and c_2 of which belong to Class 2, the entropy

$E(T) = -(c_1/n)\log_2(c_1/n) - (c_2/n)\log_2(c_2/n)$

approaches 0 the more instances belong to one class and reaches its maximum value at 1 when the instances are evenly split between the two classes. The entropy of a sample, which can be computed for an arbitrary number of classes, is the inverse of its purity.

A feature test on a training sample partitions it into a number of sets. In considering a feature test, the entropy of each set of the partition is measured, and all such measures are summed, weighted by the fraction of the original instances going to each set. The difference between the entropy of the original sample and this sum, a non-negative number, is often called the *information gain*. ID3 chooses the feature test which yields the greatest information gain.

References

1. Bareiss, R., Porter, B. and Weir, C. PROTOS: An Exemplar-based Learning Apprentice. Tech. Rept. AI87-53, University of Texas at Austin, 1988.

2. Bocionek, S. & Sassin, M. Dialog-based Learning (DBL) for Adaptive Interface Agents and Programming-by-demonstration Systems. Tech. Rept. CMU-CS-93-175, Carnegie Mellon University, 1993.

3. Caruana, R. & Freitag, D. Greedy Attribute Selection. to appear in the Proceedings of the Eleventh International Conference on Machine Learning.

4. Dent, L., Boticario, J., McDermott, J., Mitchell, T. and Zabowski, D. A Personal Learning Apprentice. Proceedings of the International Joint Conference on Artificial Intelligence, July, 1992.

5. Foltz, P.W. & Dumais, T. "Personalized Information Delivery: An Analysis of Information Filtering Methods". *Communications of the ACM 35*, 12 (1990), 51-60.

6. Holte, R. & Drummond, C. A Learning Apprentice. Tech. Rept. Unpublished., University of Ottawa, 1994.

7. Jourdan, J., Dent, L., McDermott, J., Mitchell, T., and Zabowski, D. Interfaces that Learn: A Learning Apprentice for Calendar Management. Tech. Rept. CMU-CS-91-135, Carnegie Mellon University, 1991.

8. Kay, A. "Computer Software". *Scientific American 251*, 3 (1984), 53-59.

9. Kodratoff, Y., and Tecuci, G. DISCIPLE : An Iterative Approach to Learning Apprentice Systems. Tech. Rept. UPS-293, Laboratoire de Recherche en Informatique, Universite de PARIS-SUD, 1986.

10. Kozierok, R. & Maes, P. Intelligent Groupware for Scheduling Meetings. Submitted to CSCW-92, 1992.

11. Mitchell, T.M., Mahadevan, S., and L. Steinberg. LEAP: A Learning Apprentice for VLSI Design. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, August, 1985.

12. Nakauchi, Y., Okada, T., and Anzai, Y. Groupware that Learns. Proceedings of the IEEE Pacific Rim Communications, Computers and Signal Processing, May, 1991.

13. Negroponte, N. *The Architecture Machine: Towards a More Human Environment.* MIT Press, 1970.

14. Quinlan, J.R. "Induction of Decision Trees". *Machine Learning 1*, 1 (1986), 81-106.

15. Quinlan, J.R. Generating Production Rules from Decision Trees. Proceedings of the International Joint Conference on Artificial Intelligence, August, 1987.

16. Rumelhart,D. E.,G. E. Hinton, & R. J. Williams. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing.*, Rumelhart, D. E., G. L. McClelland, & the PDP Research Group, Ed., MIT Press, 1986, pp. 318-362.

17. Russell, S. *Analogical and Inductive Reasoning*. Ph.D. Th., Computer Science Department, 1986.

18. Sadeh N. *Look-ahead techniques for micro-opportunistic job shop scheduling*. Ph.D. Th., Robotics Institute, Carnegie Mellon University, 1991.

19. Salton, G. & Buckley, C. "Improving Retreival Performance by relevance Feedback". *JASIS 41* (1990), 288-297.

20. Schlimmer, J.C. Redstone: A Learning Apprentice of Text Transformations for Cataloging. Unpublished.

21. Sheth, B. & Maes, P. Evolving Agents for Personalized Information Filtering. Proceedings of the Ninth IEEE Conference on AI for Applications, 1993.