Calendar Assistants That Learn Preferences

Jean Oh and Stephen F. Smith
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{*jeanoh,sfs*}*@cs.cmu.edu*

Abstract

Calendar scheduling is a personal behavior and there are diverse factors on which the user's decision depends. Whether the user is initiating a new meeting or responding to a meeting request she chooses an action with multiple objectives. For instance, when trying to schedule a new meeting at a preferred time and location, the user may also want to minimize change to her existing meetings, and she takes a scheduling action that best compromises the overall objectives. Our goal is to build an agent that can predict the best scheduling action to take, where "best" is defined in terms of the user's true preference. We take a machine learning approach and focus on the problem of learning the user's preference, through observation of the user as she engages in meeting scheduling episodes. We propose a hybrid preference learning framework in which we first learn utility functions of simple individual preferences such as preferred time-of-day, and then qualitatively evaluate complex scheduling options by learning a classifier from pairwise preferences. We summarize proof of principal experiments that illustrate both types of learning.

Introduction

There has been growing interest in creating software agents that can assist users with daily routine tasks. Our particular focus in this context is automated calendar scheduling. There exist several commercial software calendar programs (e.g., Microsoft Outlook Exchange) that support some form of basic meeting scheduling protocol. However, for the most part these are bookkeeping systems, with some basic capabilities for finding common free slots in shared calendars. Except for very simple forms of start and end time preferences, there is no ability to model and incorporate user scheduling preferences. This is a fundamental limitation since fine-grain customization is critical to the acceptance of automated assistants in practical contexts.

In this paper, we consider the problem of incorporating user scheduling preferences into a calendar scheduling program. We take as our starting point the CMRadar system (Modi *et al.* 2004), a distributed calendar scheduling system wherein individual CMRadar agents assume responsibility for managing different user's calendars and negotiate with other CMRadar agents to schedule meetings on their users' behalf. Our specific focus is on acquiring the user preferences that should drive this meeting scheduling process. Following revealed preference theory (Samuelson 1948), our general hypothesis is that this knowledge can be learned by observing the user engage in a series of meeting scheduling episodes with other meeting participants. We propose a hybrid preference learning framework for accomplishing this, which at the lower level learns utility functions for individual simple preferences and the higher level learns a classifier for integrating simple preferences to evaluate scheduling options.

The problem of modeling and learning user preferences in calendar scheduling has received some previous attention. One of the earliest efforts on learning user preferences in the calendar scheduling problem was made by Tom Mitchell in his Calendar Apprentice (CAP) (Dent *et al.* 1992)(Mitchell *et al.* 1994). CAP used decision tree learning to suggest specific values for the attributes of meeting. In contrast our approach (and CMRadar's basic representation) start from specifications of simple user preferences as utility functions. Sen et al. (Sen, Haynes, & Arora 1997) applied voting theory to their distributed scheduling agent system, which tries to compromise conflicting user preferences during the negotiation process. In their work preferences are associated with utility values similar to our approach, but the user is responsible for specifying the quantitative data manually. PCalM (Berry *et al.* 2004) is another system that learns an evaluation function using large margin method and Naive Bayesian approach with additional active learning strategy. However, no experimental results have yet been reported in the literature.

Learning from pairwise preferences (our approach to learning a classifier for evaluating scheduling options) has been successfully used in document ranking (Cohen, Schapire, & Singer 1998) and other domains (Gervasio, Iba, & Langley 1999). In (Cohen, Schapire, & Singer 1998) the utility of features is presumed to be well defined and given to the learner, whereas we consider acquisition of utility as part of the learning problem. We also take a more general classifier approach, rather than limiting the ranking function to a linear combination of features. Finally, PLIANT (Gervasio *et al.* 2005) is an adaptive learning system in an open calendaring system that suggests candidate schedules and learns preferences from the user's selection. They used a support vector machine (SVM) in learning from pairwise preferences in which they added pairwise preferences as constraints to a quadratic optimization function.

In the next section, we attempt to frame the overall problem of learning user preferences for calendar scheduling, and describe our hybrid preference learning framework. We then summarize two proof-of-principle experiments. The first focuses on learning static, time-of-day (TOD) preferences and provides an example of the learning of utility curves that takes place at the lower level of our hybrid model. The second considers the higher level issue of integrating individual preferences to evaluate scheduling options, in this case learning a simple classifier for evaluating different meeting bumping options. Finally, in the last section, we summarize and identify areas of future research.

Problem and Approach

Calendar scheduling has several distinguishing characteristics. First, it is *incremental* and *continuous*. As opposed to reaching for a set of definite final goal states, scheduling is done in more or less a greedy fashion. When faced with a meeting scheduling decision at a given point in time, the user typically chooses the option that looks best in this context. Time inconsistent preferences have been studied by the economists (Strotz 1955) since mid '50s in which sophisticated users speculate expected future utility in order to choose an option that may turn out better in the future. Thus it is possible to make decisions that may look less promising by the measure of immediate utility value on the expectation of future events (e.g., saving a favorite timeslot for an important tentative meeting that has not yet been scheduled). For our purposes in this paper, however, we consider the order of incoming meeting requests to be random and leave this sort of look-ahead reasoning to future work. Specifically, we assume that calendar scheduling is carried out in a greedy fashion, with the goal of making decisions that locally optimize preferences with respect to the current state of the calendar.

A second distinguishing characteristic of calendar scheduling is that it is a complex personal behavior. The user makes scheduling decisions considering diverse factors. Whether the user is initiating a new meeting or responding to a meeting request she chooses an action with multiple objectives in mind. For instance, when trying to schedule a new meeting at preferred times and locations, the user may also want to minimize change to her existing meetings, and she takes a scheduling action that best compromises these objectives. The user also takes into consideration the social relation to other meeting attendees, e.g., she is willing to sacrifice her own preference if it conflicts with a higher ranked individual's preference. To provide a basis for reasoning about such diverse factors, we assume that the calendar scheduling agent has access to a set of simple preferences. A given preference has an associated utility function, which describes its desirability in different circumstances. At the lower level of our framework, the learning goal is to acquire the user's true utility values.

Of course to assess various meeting scheduling options in a given decision context, it is necessary to integrate simple

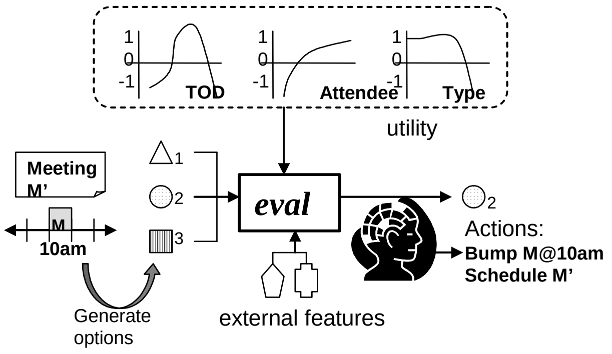

Figure 1: A model of evaluating scheduling options

preferences (and make tradeoffs where appropriate). To this end, we assume that the user has a *generic* evaluation mechanism, $eval$, which is used to rank options in a consistent way (Figure 1). Specifically , we assume that $eval$ takes all simple preferences relevant to deciding between a given pair of options as input, as well as any external features that might impact the decision at hand (e.g., an emergency or severe weather conditions), and implements a relative relation, *better than*, which can be used to order these options. Though we assume simple preferences are modeled as utility functions, we make no special assumptions about overall functional form (e.g., that $eval$ is a weighted sum of the utility values of relevant preferences). Instead, we take a more general, classifier approach to specifying $eval$. Determination of this classifier (or ranking function) is the learning goal at the higher level of our framework.

As indicated earlier, we assume that knowledge relating to user preference can be acquired by observing the user's scheduling actions over time. For instance, if the user proposes a timeslot among multiple available timeslots we assume that the proposed timeslot is preferred over other available timeslots. We define such preference over timeslots as a Time-of-Day (TOD) preference. Using similar intuition when the user cancels an existing meeting to accommodate a new meeting it is implied that the new meeting has more importance to the user than the existing one. We refer to this preemption action as *bumping*, and a user's predisposition toward bumping a given meeting as the user's bumping preference. Whereas a TOD preference can be represented as a *static* utility function involving a single attribute, timeslot, the utility of a user's bumping preference is composed of multiple features, such as meeting attendees and meeting types, and its composite utility can only be computed *dynamically* at runtime. As such, the learning of a TOD preference is a lower level problem of acquiring an appropriate utility function while the learning of a bumping preference requires formulation of an appropriate classifier.

In practice, we can get benefits from available domain knowledge. Some of this *a priori* information can come directly from the user or from central Knowledge Base (KB).

Although it is difficult to directly elicit complex preferences, the user may be willing to provide a rough approximation about some of her preferences, e.g., a typical morning or afternoon person. Similarly, information in an organization chart may be a useful a priori approximation of meeting attendee importance. We can take any initially available information as a *prior* knowledge and then update our model as appropriate through a learning process.

We make specific assumptions about the types of information that can be observed during a meeting scheduling episode and the organizational setting in which meeting scheduling takes place. In particular, we assume that a learning software agent has access to the following information when observing the user schedule meetings: (1) the user's current calendar, (2) incoming and outgoing meeting templates (Figure 2)(initiator, proposed time slots), (3) user replies (accept or refuse), (4) confirmed meeting time slots, and (5) bumped meeting templates (reschedule or cancel). We further assume that meeting scheduling takes place within a hierarchical organization where users use a common negotiation strategy when scheduling meetings that favors the preferences of higher ranked individuals. To support learning of complex preferences, the learning agent also generates a set of possible options that could have been alternatives to any actual action taken by the user. Each alternative option is paired with the user's option along with their binary relative relation *preferred* and collected as a training example for the classifier.

As a first step toward realizing our hybrid preference learning framework, we have conducted proof of principle experiments at both levels, first considering the learning of a utility curve reflecting a user's static TOD preference, and then considering the learning of a classifier that expresses a simple bumping preference. These experiments are described in the following sections.

Learning a Simple Preference

As just indicated, our approach of learning scheduling preferences first decomposes complex decision criteria into a set of *simple* preferences whose utility function is isolated on a single attribute. Time-of-day (TOD) preference is a simple preference in which the utility is defined as the user's preference over timeslots of a day horizon.

We take a statistical method in learning the utility of simple preferences. The agent's observation provides potentially noisy training examples to the learner since they are in fact results from complex preferences which are unknown to the agent. Thus, the key technique in our approach is to mitigate noise by tailoring raw training data using a priori domain knowledge. As a showcase example we describe how we learn a utility function of a simple TOD preference in the following subsections.

```
(template
          (meeting-id "MT5") (msg-id "MGS050131")
          (timestamp "2005-1-31 [15:04 -0500]")
          (initiator "sfs@cs.cmu.edu")
          (duration 3600) (location "NSH1305")
          (time-slots
                    (time-slot
                              (earliest-start-time "2005-01-31 [15:00 -0500]")
                              (latest-finish-time "2005-01-31 [16:00 -0500]")
                              (start-time "2005-01-31 [15:00 -0500]")
                              (finish-time "2005-01-31 [16:00 -0500]")
                              (priority 0.8)
                              (status "confirmed")))
(attendants
                    (attendant (id "sfs@cs.cmu.edu") (level 1.0))
                    (attendant (id "mmv@cs.cmu.edu") (level 1.0)))
(purposes
            (purpose (predefined-kind ":project-meeting")
                     (description "CMRADAR project meeting") (special-note "nil"))))
```

Figure 2: An example of CMRadar Meeting Template

Basic Modeling Assumptions

A Template shown in Figure 2 is a meta language that describes a meeting which is used as the communication medium among the subcomponents within CMRadar. The template is a stateless representation of a meeting at a fixed time point, thus the message types, e.g., whether it is an initial request or a reply to an existing request, are only implicitly defined. For purposes of this paper, we use the following notation to distinctly refer to the specific types of messages although they all share the same internal representation, i.e. Template.

- We define a calendar to be a sequence of time slots of equal duration over some horizon. Let C_u be the calendar of user u, and $C_u(t)$ refer to time slot (t) of C_u.
- A meeting request $Req_{i,A,T}$ is initiated by an initiator i, and designates a set of attendees A and a set of one or more proposed time slots T.
- A reply or response to meeting request r by user u is designated $Resp_{u,r}$ and specifies a value of either $accept$ or $refuse$ for each time slot $t \in T$.
- A Scheduled Meeting $M_{i,A,t}$ similarly designates an Initiator i, a set of attendees A and a specific time slot t. For all $u \in A$, $C_u(t) = M_{i,A,t}$.
- A *static* user meeting time preference model is expressed as a utility curve over some sequence of time slots. Preference curve denoted by $Pref_u$ determines the relative desirability of different $available$ time slots.

From the standpoint of the learning agent, the goal is to observe user u in the process of scheduling meetings and to acquire u's preference curve $Pref_u$. We assume that the learner sees each meeting request $Req_{i,A,T}$ involving u, each $Resp_{u,r}$ involving u, and receives confirmation of each scheduled meeting $M_{i,A,t}$ involving u. The learner also has access to u's current calendar at all times. Finally, the learner has knowledge of the ranks of all individuals in the organization, and assumes that all individuals use a common strategy for negotiating meeting times in which the preferences of higher ranked individuals are favored.

Our hypothesis is that under these assumptions, accumulation of the above meeting information over some number of user scheduling episodes is sufficient to enable the agent to learn the user's true meeting time preference. To test this hypothesis we use a set of CMRadar scheduling agents to simulate meeting scheduling under the above assumptions and generate training data for the learning agent. We then evaluate the ability of the learning agent to learn the true preference model of a given CMRadar agent.

Learning TOD Preference

We take a statistical approach to learning a static TOD preference curve for a given user from observed meeting scheduling data. Conceptually, our approach views user's meeting scheduling actions and results as *noisy* examples of the user's underlying preference model (both positive and negative). The key point of our approach concerns how to weight these votes to minimize the noise.

In more detail, the following information is collected as the user engages in meeting scheduling:

- $InitPropCt_u(t)$: accumulates TOD Time slots proposed by the user when initiating a meeting. The potential obscuring factor (or source of noise) in this data, however, is the density of C_u; if the most preferred time slots are already occupied, then less preferred time slots will necessarily be proposed.

  $$InitPropCt_u(t) = InitPropCt_u(t) + (1 - Density_{C_u})$$

  where

  $$Density_{C_u} = \frac{OccupiedSlots_{C_u}}{TotalSlots_{C_u}}.$$

  In other words, the evidence for a given proposed TOD time slot is discounted by the current density of C_u.

- $RefusedCt_u(t)$: accumulates TOD Time slots that are available but refused by the user when responding to a meeting request. In this case, the user's response provides active negative evidence for the time slot(s) in question.

  $$RefusedCt_u(t) = RefusedCt_u(t) + 1.$$

- $ConfirmedCt_u(r,t)$: accumulates TOD Time slots of confirmed time slots for meetings that are initiated by r. In addition to $Density_{C_u}$, here there is also a second source of noise relating to the relative ranks of meeting attendees in the organization. Taking into account the common negotiation protocol that favors higher ranked individuals, we assume that the user will tend to reveal more truthful preferences when negotiating with lower ranked individuals. To account for this, evidence relating to confirmed meeting times is differentiated by the rank of the meeting initiator.

  $$ConfirmedCt_u(r,t) = ConfirmedCt_u(r,t) + (1 - Density_{C_u}).$$

Using the above computations, we collect "votes" for each TOD time slot. We then use the weighted k-nearest neighbor (kNN) algorithm (Fix & Hodges 1951) to consolidate this data and smooth the curve. The basic idea here is to predict the utility value for a given TOD time slot using *k* similar data points in the training set. Here similarity is defined as a combination of both distance between TOD time slots and the distance between a meeting initiator's rank and the user's rank in the organization.

In more detail, the learned user preference model is computed according to the following four step procedure:

1. **Integrate TOD time slot values in** $ConfirmedCt_u$ - Taking the user's rank R_u into account, weighted kNN is applied to average the values accumulated for each TOD time slot (i.e., each column in the matrix). The result of this smoothing step is a flattened vector of Confirmed meeting votes, designated $FlatConfirmedCt_u$.
2. **Combine collected data** - Compose final "votes" for each TOD time slot as follows:

   $$TS_u = w_1 \times InitPropCt_u + w_2 \times FlatConfirmedCt_u - w_3 \times RefusedCt_u$$

3. **Smooth adjacent time slot data** - Weighted kNN is applied again, this time to the consolidated TOD time slot vector TS_u. Following the assumption that the actual (true) user preference will tend to be continuous, each TOD time slot value is averaged with the values of the k neighboring TOD time slots, discounted by TOD distance.
4. **Normalize final values** - Finally, the values in TS_u are normalized to the range $[-1, 1]$ to produce the learned preference utility curve.

Evaluation and Result

Our evaluation contrasts the performance of three preference models: **true** model, **learned** model, and **random** model where random model assigns random utility values to time slots. Each model is applied to schedule a common (new) sequence of meetings, and in each case the final resulting calendar is evaluated with respect to how well it satisfies the true user preference model. More precisely, the quality of the resulting schedule is determined as: $Q = \sum_{m \in Mtgs_u} \frac{Pref_u(TimeSlot(m))}{|MTGS_u|}$, where $MTGS_u$ is the set of meetings in C_u and $TimeSlot(m)$ is the time slot in which meeting m is scheduled.

We used a set of CMRadar agents to simulate meeting scheduling in a 4 person organization. We ran two sets of experiments for relatively simple and complex utility models. In both simple and complex cases statistical significance test showed that the learned model is significantly better than the random model. Further, the confidence intervals showed that the quality difference between the true model and the learned model is less than 0.04 with 95% confidence level in both cases, providing a very strong supporting evidence. These two experiments provide initial evidence of the ability to learn static user preference models for meeting scheduling through observation. Further details may be found in (Oh & Smith 2004)

Learning a Complex Preference

In the previous section we showed how a CMRadar agent learns utility of a decomposed simple preference through an example of TOD preference. In general many such heterogeneous preferences come into play when users make complex decisions. We define a *complex* preference to be one whose evaluation combines more than one simple preferences.

After learning individual simple preferences our next and more practical task is to combine various kinds of simple preferences together to yield an overall basis for comparing scheduling options. For example, when the user has a highly constrained schedule some amount of rescheduling is unavoidable, and she trades off various simple preferences when deciding which meeting should be bumped.

In combining lower level simple preferences we learn a binary classifier from pairwise preferences. As a proof of concept experiment of higher level learning in our hybrid approach we take the above mentioned bumping decision as an example of complex preference evaluation. We simplify our problem here by assuming that, given a set of existing meetings, the user chooses the least important meeting to bump in order to accommodate a more important meeting. We further simplify our problem by considering only the features that are defined in the meeting template (Figure 2).

Since our approach is passive learning given a user's action, the learning agent generates all alternative options, i.e., all available actions from the current state are applied and in each case the results are then paired with the user's choice. Each pair of options and their relative relation are then converted into a feature vectors. In the bumping case, the set of alternative options corresponds to a set of pre-existing meetings in the calendar and it is assumed that the bumped meeting is less preferred than all the other existing meetings.

We assume that each feature is associated with a simple preference that can be learned similarly as TOD preference. In our prototype experiment we assume that the utility functions for individual simple features have already been learned and are available to the agent. The feature vectors are translated into utility values prior to being fed to the classifier. The agent then learns a binary classifier from these accumulated training examples for purposes of ranking scheduling options when responding to new meeting requests.

In the experiment described below, a simulation was used to generate a data set of starting schedules (each abstracted simply as a set of meetings). We used a weighted sum of the feature values to model the user's true evaluation function in our simulation. It is important to note that, however, our approach is not limited to any specific evaluation mechanism. Using this generated data set we learned a binary classifier such that given two meeting templates it can determine which one of the given meetings is more important to the user. Using this binary compare operator it is trivial to compute the least important meeting.

Experimental settings

In this section we briefly describe how the data sets were generated and how the experiment was carried out. To do so, we introduce the following notation:

- A meeting M_k is represented by a set of features that are associated with M_k, where k is the meeting ID.
- Let f_j denote the j^{th} feature in the feature set. Let $f_{k,j}$ denote the value of the f_j of M_k. Suppose there are l features that are relevant to determining the priority of the meetings. Then we define M_k as $\{ f_{k,1}, f_{k,2}, ... , f_{k,l} \}$.
- The features are weighted by their importance, i.e. the weight of a feature indicates how much the feature contributes to the decision. Let w_j denote the weight of f_j feature.
- We virtually define a generic function $eval(M_k)$, which is hidden to the agent. The user is assumed to make consistent decisions according to this hidden model.
- M_x is more $bumpable$ than M_y iff. $eval(M_x) > eval(M_y)$
- When it is necessary to bump one of the existing meetings, the user chooses the most bumpable meeting based on the rationale described above.

A single bumping episode involves a set M_{ex} of m existing meetings in the calendar, and a single new meeting, M_n. Suppose that one of the existing meetings is bumped, and let M_b denote the bumped meeting. This implies that $M_b > M_n$ and $M_b \geq M_k$ for all $M_k \in \{M_{ex} - M_b\}$. (Note that the latter relation includes equality, thus introducing a potential source of noise in the data.) Each pairwise comparison provides one classification example, e.g., given two sets of features it produces a binary output, *to bump or not to bump*.

We also add the converse samples to the training examples as well, e.g., $M_n < M_b$. Although it may look redundant, it is in fact necessary to include these twin examples, since the classifiers are not informed about the symmetry of the two sets of features. In the case that M_n fails to bump any of the existing meetings, meaning that M_n was the most bumpable meeting, the training data is generated similarly.

The test phase also follows a similar process of data generation. Given a new meeting M_n and a set of existing meetings, M_{ex}, we now use the trained classifiers in order to compare M_n and the meetings in M_{ex}. When there are m alternatives, handling one bumping case requires m classifications to determine the most bumpable meeting.

We fixed the size of test data set to 35 (corresponding to 35 bumping episodes). Three evaluation measures were computed: classification error rate, exact match rate, and exact match plus partial match rate. The exact match means that the agent's decision was exactly same as the answer (the user's preferred decision). There are cases in which the alternative options are in fact not much different from one another; in such a case, if the agent picked a meeting that did not exactly match with the user's choice but was still $reasonable$, the agent receives some partial credit (this is the third performance measure). We define that the choice is $reasonable$ if the difference in probability of being bumped of the two meetings using the true evaluation metric is less than 0.1.

Classifiers

A classifier takes a vector of feature values as input and predicts which class the given input belongs to. It is assumed that there is a predefined set of classes for categorization. In our general model, there are two such classes: preferred or not preferred. In our simplified Bumping preference experiment the question is rephrased as *to bump or not to bump*.

To determine the best performing classifier in our scheduling domain we conducted our experiment with a number of different schemes. This section briefly describes the classifiers that we tested in the experiments.

- Naive Bayes (NB) Classifier: We used a Gaussian Bayes classifier which assumes Gaussian distribution of data. NB also assumes that all the features are independent from one another. This naive assumption, however, becomes an advantage when dealing with a large number of features.
- Joint Bayes (JB) Classifier: JB classifiers use covariance matrix to eliminate the feature independence assumption of NB. JB performance is in general very competitive despite the simplicity of algorithm. The covariance hinders its performance as the number of features grows.
- k-Nearest-Neighbor (kNN): kNN uses k similar data points to predict the class of a new input. We use a simple Euclidean distance to define similarity. In the case of a tie, one more neighbor, the $(k+1)^{th}$ nearest one, is added if k is less than half the size of the training set. Otherwise, the k^{th} neighbor is dropped.
- Support Vector Machine (SVM): SVM is considered the best available classifier in many domains and the idea is quite different from other classifiers in the fact that it uses only the data points near the class boundary. We used SVM-KM Toolbox (Canu, Grandvalet, & Rakotomamonjy 2003). This implementation of the SVM classifier uses quadratic programming. There was no bound on the Lagrangian multipliers, and lambda was 0.000001. We did not investigate the effects of these parameters on the SVM's performance in our experiments.
- Decision Trees (DT): We used DT available in MAT Labs toolbox. The default criterion for choosing a split used in MAT Labs treefit function is Gini's Diversity Index (GDI), which is what we used here.
- Random Classifier: The random classifier determines the class completely randomly. We included this model to use it as a baseline.

Experimental Results

In our experiment we tested with both discrete and continuous feature values. Discrete features represent binary features whose value is directly mapped to the exact binary utility values, e.g., 1 if the feature exists, 0 otherwise. We varied the number of features from 1 to 10 in the case of continuous features, and up to 30 for the discrete features.

Figure 3 shows the learning curves for the 6 classifiers tested for the case of 5 continuous features. It shows that all the classifiers start to converge after seeing about 10 bumping examples. SVM and Joint Bayes performed the best, and their error rates after convergence were below 0.1. Decision Tree showed rather disappointing results, with an error rate of around 0.2.

Overall, SVM outperformed all the other classifiers. The Gaussian Bayes classifiers' performances were quite close to SVM with the continuous features. Whereas Nave Bayes consistently performed relatively well, the Joint Bayes classifier's performance is quickly degraded as the number of features increased, especially with the discrete features.

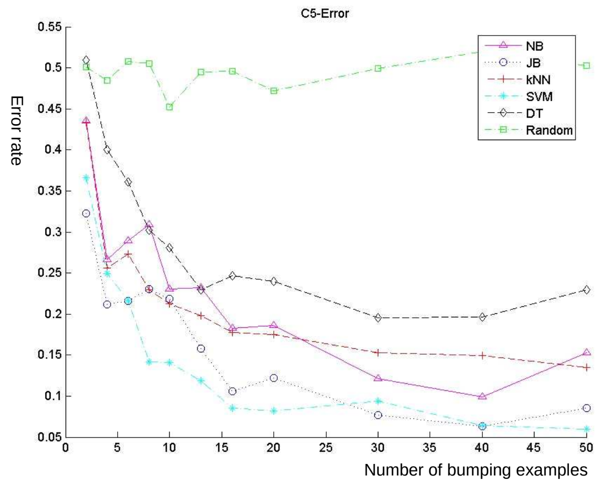

Figure 3: Learning curves using various classifiers (5 continuous features)

Both JB and SVM classifiers' performance reached above .90 for the exact match rate, and above .95 with partial credits, for up to 10 features. This means that after observing 10-15 bumping episodes the agent did exactly what the user would do 90% of the time, and for half of the other times, it still made reasonably good choices.

Conclusion and Future Work

Calendar scheduling is a complex decision making problem. The user reasons about various mixtures of preferences that are directly or indirectly related to the scheduling problem. In this paper we proposed a hybrid approach of learning utility functions of simple preferences and then learning a classifier to combine them at a higher level to evaluate complex scheduling options.

We assume that the user makes decisions that produces the best combined expected utility. We also assume that we can learn the utility functions of individual factors that affect the user's decision by observing a series of scheduling episodes. Our evaluation model then uses the learned utility of all those factors to compare alternative options and choose the best. To accomplish this, we proposed using a classifier to perform as a compare operator.

As a proof of concept of our approach we have described two separate experiments. First, we demonstrated an approach to learning the utility function that characterizes a user's simple TOD preference. In this initial experiment the quality of the learned model was found to be as good as the true model with quality difference less than 0.04 with 95% confidence level. Second, we demonstrated the capability to learn a binary compare operator for higher level evaluation of bumping options. In our prototype experiment the agent achieves over 90% accurate prediction rate after observing only 10-15 bumping examples.

We are currently implementing a generalized preference evaluation framework in our CMRadar scheduling agent to accomplish more thorough evaluation of our approach. Our current model flattens all the features into a vector on purpose in order to suit various types of user's ill-structured preferences. We intend to also evaluate other possibilities of recovering more structured preference models.

There are still many interesting future research issues to think about. We aim to learn more sophisticated types of preferences, e.g., how the user's scheduling behavior changes as it gets closer to certain events such as deadlines. Also certain dynamic effects change the utility of TOD preference, e.g., whether the user prefers to schedule meetings back to back (Back-to-Back preference). Similarly, preferred start or end times of a day may also be dynamically adjusted by the meetings that are scheduled unusually early or late, which changes the TOD utility of that specific day.

Consideration of room reservations for scheduled meetings adds an extra dimension of complexity to our existing calendar scheduling domain. Finally, it is also interesting how external features such as emergency cases should be incorporated into the user's scheduling decision. Our optimization continues to be greedy since we do not speculate about future meetings. But over time we may be able to learn potential future meetings in consideration, e.g., implicit recurring meetings.

Acknowledgement

We would like to thank Simon Fung for helpful discussions and his work on evaluation of SVM and Decision Trees. This work was supported in part by the Department of Defense Advanced Research Projects Agency (DARPA) under contract #NBCHC030029.

References

Berry, P.; Gervasio, M.; Uribe, T. E.; Myers, K.; and Nitz, K. 2004. A Personalized Calendar Assistant. *AAAI Spring Symposium Series, March*.

Canu, S.; Grandvalet, Y.; and Rakotomamonjy, A. 2003. Svm and kernel methods matlab toolbox. Perception Systmes et Information, INSA de Rouen, Rouen, France.

Cohen, W. W.; Schapire, R. E.; and Singer, Y. 1998. Learning to order things. In Jordan, M. I.; Kearns, M. J.; and Solla, S. A., eds., *Advances in Neural Information Processing Systems*, volume 10. The MIT Press.

Dent, C. L.; Boticario, J.; McDermott, J. P.; Mitchell, T. M.; and Zabowski, D. 1992. A Personal Learning Apprentice. In *Proceedings of AAAI-92*, 96–103.

Fix, E., and Hodges, J. L. 1951. Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties. In *Technical Report Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Randolf Field, Texas*.

Gervasio, M. T.; Moffitt, M. D.; Pollack, M. E.; Taylor, J. M.; and Uribe, T. E. 2005. Active preference learning for personalized calendar scheduling assistance. In *Proceedings of the 2005 International Conference on Intelligent User Interfaces*.

Gervasio, M.; Iba, W.; and Langley, P. 1999. Learning user evaluation functions for adaptive scheduling assistance. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 152–161.

Mitchell, T. M.; Caruana, R.; Freitag, D.; McDermott, J. P.; and Zabowski, D. 1994. Experience with a Learning Personal Assistant. *Communications of the ACM* 80–91.

Modi, P. J.; Veloso, M.; Smith, S. F.; and Oh, J. 2004. CMRadar: A Personal Assistant Agent for Calendar Management. In 6^{th} *International Workshop on Agent-Oriented Information Systems (AOIS)*, 134–148.

Oh, J., and Smith, S. F. 2004. Learning User Preferences in Distributed Calendar Scheduling. In *The International Series of Conferences on the Practice and Theory of Automated Timetabling*, 35–49.

Samuelson, P. A. 1948. Consumption Theory in Terms of Revealed Preference. *Econometrica* 15:243–253.

Sen, S.; Haynes, T.; and Arora, N. 1997. Satisfying user preferences while negotiating meetings. *International Journal of Human-Computer Studies* 47:407–427.

Strotz, R. H. 1955. Myopia and Inconsistency in Dynamic Utility Maximization. *Review of Economic Studies* 23:165–180.

Wasserman, L. A. 2004. *Bayesian Inference*. Springer. chapter 11.