

Explaining Preference Learning

Alyssa Glass

CS229 Final Project

Stanford University, Stanford, CA

glass@cs.stanford.edu

Introduction

Although there is existing work on learning user preferences in various systems, the outputs of such systems tend to be confusing to users. Studies of users interacting with such systems show that when the system incorrectly predicts a user's preference, the user may attempt to correct the views of the system (for instance, by randomly providing more training examples). In many cases, however, users lose patience with this approach, largely due to a misunderstanding of how the underlying system is using the data to make predictions. Ultimately, they may stop trusting the system entirely. Even when the system is correct, users view such outcomes as "magical" in some way, but are unable to understand why a particular answer is correct, or whether the system is likely to be helpful in the future.

In this paper, we describe the augmentation of a preference learner, designed to provide meaningful feedback to the user in the form of explanations. We begin by describing the preference learning problem in the specific context of semi-autonomous meeting scheduling, and the preference learning system currently being used to solve the problem in the CALO research project. We describe current research on usability in the face of active learning systems, including a study that we conducted on learning and trust. We then describe how this preference learning system was modified to store meta-information about its learning during the course of system execution, and how this meta-information is used to provide explanations of answers concluded by the system. Finally, we show how these explanations can be incorporated into the larger scheduling system to provide transparency into the learning system, enabling trust between user and system.

Motivation

The problem of scheduling meetings among groups of busy, over-committed professionals is a well-known problem for anyone who has worked in an office environment. Often, a series of emails or phone calls among desired meeting participants are required before a jointly acceptable time has been chosen. Central to this problem is the incorporation of preferences – different participants may prefer particular meeting lengths, particular times of the day, particular days of the week, etc.

Complicating the problem of incorporating user preferences is the problem of dealing with conflicts. The PTIME personalized scheduling assistant (Berry, et al. 2006) is one system built to automatically handle these scheduling problems while taking into account user preferences, as part of the Cognitive Assistant that Learns and Organizes (CALO) project.

In PTIME, a user initializes the system by indicating some initial preferences for meeting scheduling (for instance, preferring morning vs. afternoon meetings; whether to overlap meetings if necessary; and whether to drop individual meeting participants in order to fit a schedule) as well as information on how these preferences should be traded off with each other when they conflict (for instance, whether a meeting should be shortened in order to not overlap another existing meeting). To schedule a meeting, a user enters both broad constraints (such as “any time before next Thursday”) as well as some suggested relaxations (such as “Joe is an optional participant”). PTIME then attempts to solve the constraint problem defined by the meeting specification entered by the user and the preferences of the user and other meeting participants, presenting the user with multiple possible schedules based on these constraints, some of which may *relax* the constraints along varying problem dimensions. These possible schedules are presented to the user roughly in preference order. (In some cases, less-preferred schedules are presented ahead of other schedules in order to provide the user with more variety of options and to allow the preference learner to better explore the space; details can be found in (Yorke-Smith et al. 2006).) When the user then selects one of the provided schedules, the system’s model of the user’s preferences is updated.

This update of the user’s preferences is computed by PLIANT (Preference Learning through Interactive Advisable Nonintrusive Training) (Gervasio et al. 2005). PLIANT uses support vector machines (SVMs) to update the preference model, using each selection of a schedule as a new data point, as described in the next section.

Although PTIME, with on-line learning computed by PLIANT, has been deployed in restricted settings and used by several users for accomplishing limited tasks, adoption of the system has been slow. A study of PTIME reported in (Yorke-Smith et al. 2006) and a larger CALO-wide user study conducted by us, in conjunction with Deborah McGuinness and Michael Wolverton, both confirm that a

significant barrier to the wide adoption of a system like PTIME is the problem of transparency. Most users simply do not trust a system that is constantly modifying itself through learning unless they are able to ask questions and be provided with information about the inner workings of the system. PTIME has begun to address this problem by explicitly indicating where constraints are being violated, but much more work is needed to provide explanations of the preference learning. In particular, since users view the ranked schedules provided by PTIME as suggestions for which schedules the user is expected to prefer, explanations providing transparency into why particular schedules are recommended over others is vitally needed for the usability of the system. To solve this problem, then, we attempt to provide transparency into the PLIANT learning system.

Active Preference Learning

Active learning in PLIANT relies on a starting feature set of 6 criteria, initialized through input from the user. The criteria are:

1. Scheduling windows for requested meeting
2. Duration of meeting
3. Overlaps and conflicts
4. Location of meeting
5. Participants in meeting
6. Preferences of other meeting participants

The first time PTIME is started by a user, the system elicits initial preferences for these six criteria through a wizard-type interface. For example, for “overlaps and conflicts,” the system asks the user whether he prefers meetings to never overlap, or to allow the system to “double-book” him in two meetings at the same time when necessary. The system also asks the user to rank the importance of these six criteria relative to each other; for example, if the user’s preference on duration and location cannot both be met, the system stores pair-wise information about which criteria is more important.

Given the partial utility functions implied by each of these criteria, PLIANT combines these utility functions into a single function using a 2-order Choquet integral:

$$F(z_1, \dots, z_n) = \sum_i a_i z_i + \sum_{i,j} a_{ij} (z_i \wedge z_j)$$

where each $z_i = u_i(x_i)$, the utility for criteria i based on value x_i . (Note that this is essentially a function over schedules, not individual meetings; for a given schedule, each x_i is a measure of the degree to which the full schedule, containing multiple meetings, matches the given criteria.) Thus, the learning task for PLIANT is to learn the weights a_i and a_{ij} associated with each individual and pair-wise matching of the six criteria. Justification for the use of a 2-order Choquet integral is in (Yorke-Smith et al. 2006).

This function is then combined with the initial preferences elicited from the user by the full evaluation function:

$$F^*(Z) = \alpha \times A \cdot Z + (1-\alpha) \times W \cdot Z$$

where Z is the full schedule being evaluated, W is the full set of a_i and a_{ij} weights learned above, A is the initial set of preferences elicited from the user, and α is a parameter indicating the impact of the elicited versus learned preferences. The α parameter is decayed over time, both to accommodate a user’s changing preferences and to acknowledge the difficulty many users have with explicitly indicating their own scheduling preferences without the use of examples.

The overall system, then, has the following work flow: the system elicits initial preferences from the user (the A vector above) once, during PTIME’s first use. Then, for each new meeting, the user specifies meeting parameters and constraints. The PTIME constraint solver generates several candidate schedules (Z ’s), relaxing constraints when necessary, according to the stated constraints and the user’s existing calendar. The candidate schedules are presented to the user in (roughly) the calculated preference order, and the user selects a single schedule, Z . The user’s preferences (the a_i and a_{ij} weights) are then updated based on the user’s choice.

The data used to perform the preference update is both the chosen schedule, and the other schedules considered but rejected by the user. For example, if three schedules are presented in ranked order to the user, and the user chooses the third schedule, a partial ordering is imposed on the schedules indicating that the third (chosen) schedule is preferred to both the first and second schedule (Gervasio et al. 2005). These partial orderings are then added into the optimization problem solved by the SVM, in the same way that search engine results are ranked according to past clickthrough data in (Joachims 2002). Joachims’ *SVMlight* is used as the basis for PLIANT’s update of the preference weights. It is this update of the preference weights, and the resulting ranked ordering presented to the user, that we seek to explain.

Usability and Active Learning

As a guide for how best to explain the preferences learned by PLIANT, we considered the results of three user studies. Relevant points from each of the three studies are summarized below.

First, outside of the work done for this project, we conducted our own study of CALO usability (joint work with Deborah McGuinness and Michael Wolverton; not yet published). Focusing on PLIANT and PTIME, in particular, we found a general lack of understanding among users of how preferences are being updated, or even that they are being updated at all. Although there are many reasons why a recommendation from PLIANT may

violate the initial preferences entered by the user (for example, PLIANT may have more recent learned preferences that violate the initial preferences; PLIANT may be deliberately violating preferences in order to explore the search space; or the user himself may not have correctly indicated his true preferences, a common problem with explicit preference elicitation), users nonetheless expressed surprise and confusion whenever their true preferences were violated by the ranking of schedules presented to them. Many users commented that it seemed that the system was ignoring their preferences, leading them to view the system as untrustworthy. One user commented, “I trust [PTIME’s] accuracy, but not its judgment.” This type of complaint specifically about the use of preferences was common among many of the users we studied.

Next, we considered a study conducted by the PLIANT team, in which users were interviewed about their requirements for an adaptive scheduling system. One of the top requests made by these users was “transparency into assisted scheduling decisions (e.g., explanations of conflicts and learned preferences)” (Yorke-Smith et al. 2006). The authors continue by noting that “the preference model must be explainable to the user ... in terms of familiar, domain-relevant concepts.” This last requirement, in particular, is often a problem with attempts to understand and explain statistical learning methods, a problem that we address in our formulation.

The final study that we considered is by (Stumpf et al. 2007). This study considered explanations of statistical machine learning methods, focusing on a naïve Bayes learner and a rule-learning (classification) system, both in the context of learning the proper foldering of email. They found that rule-based explanations, taken directly from the rule-learning system, were most easily understood by users, but were not trusted to the same extent. They also found that similarity-based explanations, which users had a very hard time understanding, were nonetheless considered more “natural” by the users, and were easily trusted to an extent beyond what would be expected, given the level of understanding. These similarity-based explanations were roughly of the form, “This message is really similar to the message A in folder X because they have important words in common: <common words highlighted>.” The authors also noted that many users appreciate a less formal, non-technical style for explanations.

We posit that the understandability problem with similarity-based explanations noted in this last study is related directly to the findings in the PTIME study which found that explanations must be tied directly to domain concepts that are known to the user. In particular, users complained that the similarities identified by the naïve Bayes system seemed arbitrary; indeed, to users not familiar with statistical machine learning methods, it would be difficult to understand similarities among

dictionary-style feature vectors of several thousand words, and why some words and not others were considered important by the system. In our preference learning domain, as noted above, we are instead considering a relatively small space of features, each of which has previously been explained to the user in simple terms. In this domain, we are thus able to create similarity-based explanations which may enjoy the trust noted in the third study, while still staying in the realm of familiar domain concepts. These similarity-based explanations can be created naturally from the computation done by the SVM, as described in the next section.

Providing Transparency into Preference Learning

Starting with the PLIANT code base, we augmented the use of *SVMlight* with code to gather additional meta-information about the SVM itself. For the purposes of creating explanations, we considered the following SVM meta-information:

- The support vectors identified by the SVM
- The support vectors “nearest” to the query point
- The margin to the query point
- The average margin over all data points
- The non-support vectors (i.e., other data points) nearest to the query point
- Initial preferences elicited from the user, along with the current value of α
- The kernel transform used, if any

In the PLIANT system, the kernel is linear, so no information about the kernel transform was needed. If a non-linear kernel is used by the SVM, the problem of explaining the results becomes much more complicated; we do not address the issue here.

The next step was to represent this meta-information, along with underlying information about how SVMs learn, in a way suitable for producing justifications of the computation performed in PLIANT. We represented this information in Proof Markup Language (PML) (Pineiro da Silva, et al. 2006) because of its generic justification representations for general reasoners, the existing use of PML in the CALO system, and work we have done over the past year in expanding PML to represent the results of machine learning (Glass & McGuinness 2006).

For the representation in PML, we added logical rules for the deduction performed by an SVM, both to generate a conclusion given a query point and a data set, and to describe the resulting decision plane and space of data points. First, we added the base rule “hasRank” which has three antecedents (the initial preferences elicited from the user; the calculated weights, which indicate the support vectors in the SVM; and the new query point, which is a schedule proposed by the constraint reasoner) and results in a conclusion indicating the ranked preference for this

schedule, which is the result of the evaluation function defined above. The antecedent representing the support vectors contains an `iw:LearnedSourceUsage`, with a link to the data set used by the SVM, along with information about when this data set was generated. (See PML specification referenced at the end of this document.) The additional meta-information gathered from the SVM is represented through additional rules indicating how they are concluded. For example, “`hasCloseDataPoint`” takes the second two antecedents mentioned above, and concludes a single data point near the query point, with details about which dimensions are closest to the query. Depending on context, then, the expanded SVM can conclude as many of these “`hasCloseDataPoint`” conclusions as necessary; in our system, we limited this to 5 points because of the relatively small size of the full data set. Additional rules for each of the other meta-information types mentioned above were also added. Finally, we added a rule “`hasInitialRank`” which is similar to “`hasRank`,” but only considers the preferences initially elicited from the user, and provides a rank based solely on these initial preferences, without any learning.

The result, then, is a formal justification for why PLIANT recommended a particular schedule. Given this justification, we then created strategies for abstracting the justification for presentation to a user. Methods for abstracting justifications in PML already exist within the Inference Web (IW) infrastructure (McGuinness and Pinheiro da Silva 2004). We expanded these methods to cover justifications of the form created by our augmented SVM.

As discussed in the usability section above, we chose to focus on similarity-based explanations, taking advantage of the small feature vectors and the clean mapping into user-understandable terms. We also generated explanations in the first-person, as if the system were informally talking to the user about its observations, because of the preference for non-technical explanations noted in (Stumpf et al. 2007). The strategies that we created take a PML justification, and consider which of the above meta-information rules provides the strongest reason for presenting a given schedule to the user. Note that this selection is a bit subjective; we chose fairly straightforward heuristics for choosing a strategy, without making claims that the best possible explanation is given in all cases. Instead, our goal was simply to provide an explanation that is reasonable enough, so that the system is not immediately dismissed by the user.

Once a strategy is chosen, it generates a single explanation in English. These explanations are generated in a simple fill-in-the-blank format; as such, the explanations are domain dependent, and can occasionally be a bit awkward, but avoid the issue of full natural language generation. Examples of some of the explanations generated by the strategies:

- “This schedule closely follows the preferences you told me when we started.” (Used when α is large, or when the rank generated by “`hasInitialRank`” is particularly high.)
- “This schedule is similar to other schedules that you have chosen in the past, and I have observed your preference for *avoiding overlaps and conflicts*.” (Used when several of the nearest preferred data points are particularly close along a particular feature dimension.)
- “This schedule seems to be strongly preferred, based on your past selections, and I have observed that you often have a preference for *considering other participants’ preferences*.” (Used when the margin to the query point is particularly large, relative to the average margin. This strategy also attempts to call out which feature contributes the most to the margin, but often leaves out the last clause when no features stand out.)
- “This schedule does not closely match your preferences, but listing it here will help me to understand more about your preferences, so that I can do better in the future.” (Used when the ranks computed both by the learned weights and the initial preferences are poor, but PLIANT is exploring the space.)

Thus, to provide transparency into PLIANT’s preference learning, we envision a link into PTIME which enables users to ask questions about why a particular schedule is being recommended. The justification for a particular schedule, in PML, is stored when the ranking is generated. When a user then requests an explanation, this justification is parsed, an explanation strategy is chosen, and the resulting explanation is presented for the user. In the past, we have designed systems which are intended to engage the user in a full dialogue, enabling the user to ask follow-up questions and gather additional information as desired. Here, we instead show a system where a single explanation is generated for each schedule, with no possibility for follow-up questions; since our goal is trust in the overall preference learner, rather than trust in the recommendation of a particular schedule, we feel that the sum of all explanations over all of the generated schedules better accomplishes our goal, and fits more cleanly in the existing interface without distracting the user from their main goal of selecting a schedule. The resulting architecture, showing the main PLIANT architecture along with the explainer component described here, is shown in Figure 1.

For the purposes of this project, the integration of our expanded PLIANT system and explanation component into the larger PTIME system was shallow, and completed only to a degree to validate proof of concept. Using scheduling and preference data previously gathered during the CALO Year 3 Critical Learning Period (CLP)

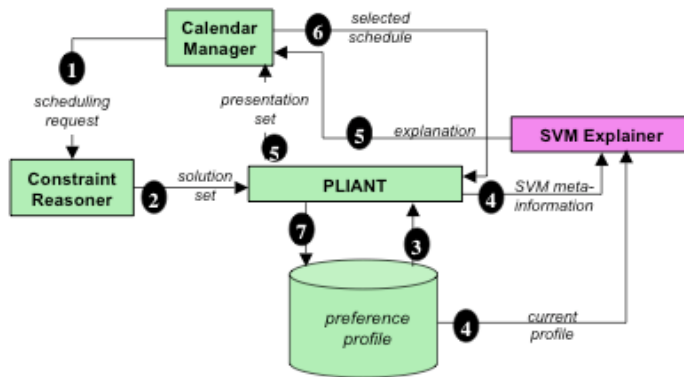


Figure 1: System architecture. Four boxes on left are the original PLIANT system. Component “SVM Explainer” has been added, and additional data flow is shown. (Original PLIANT architecture from (Gervasio et al. 2005).)

(containing scheduling and preference data for 16 users, gathered over approximately two weeks), we generated meta-information for select examples as described above, used the meta-information and PML rules as described to generate justifications for a subset of this data, and created explanations from these justifications using the strategies described above.

Conclusion

We have described the design of a system for explaining the results of an SVM active preference learner, and how it can be integrated into an adaptive scheduling assistant. We based our design on several user studies which indicate both the need for explanations of statistical machine learning systems, and the types of explanations that are required by users to build trust. Future work includes system engineering to provide a full integration into the PTIME system; analysis of whether additional meta-information or explanation strategies not studied here would also be appropriate for generating explanations; and a user study to validate the usability of the explanations being generated. We also plan to consider the extensions of this work to other SVM systems, particularly systems that are learning over larger feature sets.

Acknowledgements

We gratefully acknowledge funding support for the CALO project from the Defense Advanced Research Agency (DARPA) through contract #55-30000680 to-2 R2. We thank Melinda Gervasio, Pauline Berry, Neil Yorke-Smith, and Bart Peintner for access to the PLIANT and PTIME systems, and for helpful collaborations, partnerships, and feedback on this work. We thank Deborah McGuinness, Michael Wolverton, and Paulo Pinheiro da Silva for the IW and PML systems, and for related discussions and previous work that helped to lay the foundation for this effort. We thank Mark Gondek for access to the CALO

CLP data, and Karen Myers for related discussions, support, and ideas.

References

- Berry, P., Conley, K., Gervasio, M., Peintner, B., Uribe, T., and Yorke-Smith, N. *Deploying a Personalized Time Management Agent*. AAMAS-06 Industrial Track, pages 1564-1571, 2006.
- CALO, 2006. <http://www.ai.sri.com/project/CALO>.
- Gervasio, M.T., Moffitt, M.D., Pollack, M.E., Taylor, J.M., and Uribe, T.E. *Active Preference Learning for Personalized Calendar Scheduling Assistance*. Proceedings of Conference on Intelligent User Interfaces 2005 (IUI-05), 2005.
- Glass, A. and McGuinness, D.L. *Introspective Predicates for Explaining Task Execution in CALO*. Technical Report, KSL-06-04, Knowledge Systems, Artificial Intelligence Laboratory, Stanford University, 2006.
- Joachims, T. *Optimizing Search Engines Using Clickthrough Data*. Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD), ACM, 2002.
- McGuinness, D.L., and Pinheiro da Silva, P. 2004. *Explaining Answers from the Semantic Web: The Inference Web Approach*. Journal of Web Semantics, 1(4), 397-413. <http://iw.stanford.edu>
- Pinheiro da Silva, P., McGuinness, D.L., and Fikes, R. *A Proof Markup Language for Semantic Web Services*. Information Systems, Volume 31, Issues 4-5, June-July 2006, pp 381-395.
- PML specifications: <http://iw.stanford.edu/2006/06/pml-provenance.owl> <http://iw.stanford.edu/2006/06/pml-justification.owl>
- Stumpf, S., Rajaram, V., Li, L., Burnett, M., Dietterich, T., Sullivan, E., Drummond, R., and Herlocker, J. *Toward Harnessing User Feedback for Machine Learning*. Conference on Intelligent User Interfaces 2007 (IUI-07) (to appear).
- Yorke-Smith, N., Peintner, B., Gervasio, M., and Berry, P. M. *Balancing the Needs of Personalization and Reasoning in a User-Centric Scheduling Assistant*. Unpublished manuscript (under submission) 2006.