User-Controllable Learning of Security and Privacy Policies*†

Patrick Gage Kelley, Paul Hankes Drielsma, Norman Sadeh, Lorrie Faith Cranor
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213, USA
{pkelley, paulhd, sadeh, lorrie}@cs.cmu.edu

ABSTRACT

Studies have shown that users have great difficulty specifying their security and privacy policies in a variety of application domains. While machine learning techniques have successfully been used to refine models of user preferences, such as in recommender systems, they are generally configured as "black boxes" that take control over the entire policy and severely restrict the ways in which the user can manipulate it.

This article presents an alternative approach, referred to as *user-controllable policy learning*. It involves the incremental manipulation of policies in a context where system and user refine a common policy model. The user regularly provides feedback on decisions made based on the current policy. This feedback is used to identify (*learn*) incremental policy improvements which are presented as suggestions to the user. The user, in turn, can review these suggestions and decide which, if any, to accept. The incremental nature of the suggestions enhances usability, and because the user and the system manipulate a common policy representation, the user retains control and can still make policy modifications by hand.

Results obtained using a neighborhood search implementation of this approach are presented in the context of data derived from the deployment of a friend finder application, where users can share their locations with others, subject to privacy policies they refine over time. We present results showing policy accuracy, which averages 60% upon initial definition by our users, climbing as high as 90% using our technique.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; I.2.6 [**Artificial Intelligence**]: Learning; H.5 [**Information Interfaces and Presentation**]: Miscellaneous

General Terms

Security, Human Factors

Keywords

Security and privacy policies, usable security, user-controllable learning

*This work is supported by NSF Cyber Trust grant CNS-0627513. Additional support has been provided by Nokia, France Telecom, the CMU/Microsoft Center for Computational Thinking, and CyLab/ARO.
†*Patent pending.*

AISec'08, October 27, 2008, Alexandria, Virginia, USA.

1. INTRODUCTION

Motivations.

A broad and growing number of applications allow users to customize their policies. From the network administrator maintaining complex and verbose firewall access control lists to the Facebook user struggling to understand the semantics of the site's privacy settings, studies have consistently shown that novice and expert users alike find it difficult to effectively express and maintain such policies. In one study, for instance, test users asked to express file permission policies within the native Windows XP interface achieved accuracy rates as low as 25% [5]. This reflects a significant gap between the users' intended policies and the policies that they manage to express in concrete policy specification languages and their associated interfaces.

Given this difficulty, it is highly desirable to support users in the tasks of policy specification and maintenance, with the aim of helping them narrow this gap. While a number of machine learning applications rely on simple forms of user feedback to improve their performance (e.g. spam filters, Amazon recommendations), little work has been done to develop configurations of these techniques that support closer collaboration between machines and users. Recent studies, however, have shown that users are often willing to provide richer feedback and would benefit from richer interactions (e.g. [12]).

Overview.

In this paper, we present a user-oriented approach to policy refinement that collects feedback from the user to help identify (or *learn*) incremental improvements to her policy. The most promising improvements are presented to the user, who in turn decides whether or not to accept them (see Fig. 1). This *user-controllable policy learning*

approach contrasts with the black box configuration in which most machine learning techniques are traditionally deployed. Rather than restricting the level of control the user has over the policy model—for example, limiting the user to providing occasional feedback on system decisions—it allows the user and the system to work in tandem on a common model. By focusing on incremental changes to policies originally specified by the user, it aims to make it easier for the user to understand proposed changes and decide whether or not to accept them. At any point in time, the user can override the system and make her own changes.

This is particularly useful in situations where new conditions arise (e.g. changes in the user's social network, new types of network attacks) and where user-defined changes are more effective than waiting for the system to refine policies or relearn from scratch. In addition, allowing the user to retain control over improvements learned by the system reduces the risk of the system introducing particularly bad policy modifications. This is especially important when dealing with security and privacy policies where the ramifications of an incorrect policy decision can be quite costly, but useful in any environment where users are frustrated by a lack of control. In this paper, we assume policy models expressed as collections of condition/action rules. This model is sufficient to capture a broad range of policies, including a wide variety of security and privacy policies such as XACML policies.

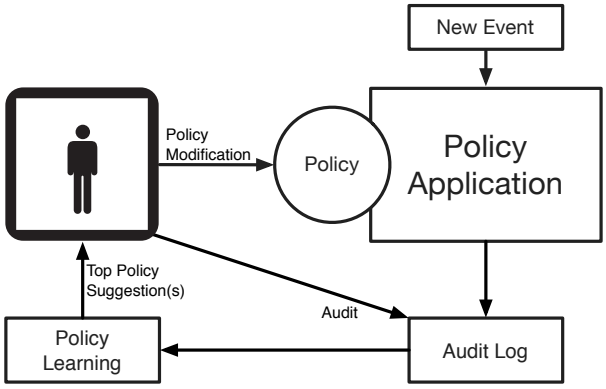

Figure 1: Usage Diagram

In addition, our approach fulfills two requirements that we consider critical to any security and privacy focused system. First, the users' interaction with such policy support systems must be simpler than their interaction with the languages used to specify the policies themselves. As a paradigm for this interaction, we propose feedback on system decisions, which has the advantage of giving the users concrete examples upon which to to base their responses.

Secondly, we require complete system transparency, so that the user maintains control over the policy and any modifications to it at all times. To ensure control over system-based changes, our solution proposes a selection of improvements, leaving control as to whether to adopt any of the suggestions in the user's hands. Research has shown that an integral component of users' perception of privacy is that they maintain control over the disclosure of their personal information [6]; we thus allow the user to make changes directly to the policy at any time, in contrast to many current recommender systems, where the user's sole interaction with the system is through provision of feedback.

PeopleFinder.

While we emphasize that our findings are general, we present them with the help of a running example, the PeopleFinder location-based social network developed at the Mobile Commerce Lab at Carnegie Mellon University [10], a screenshot of which is shown in Fig. 2. PeopleFinder allows users of location-enabled laptops and cell phones to share their location with their network of friends in a privacy-sensitive way. Privacy policies in PeopleFinder permit disclosure based on three criteria: the identity (or group membership) of the user making the request, the weekday and time of the request, and the location of the requested user. Thus, privacy policies can be comprised of rules such as "Allow disclosure to the group Co-Workers on weekdays between 9 and 5, but only when I am actually at my office." In addition, users of PeopleFinder can provide feedback on system decisions. The background window shown in Fig. 2 illustrates the feedback interface, whereby users review a history of requests for their location and indicate their level of satisfaction with the disclosure decisions made by the system. Users can also ask for additional details about the requests and obtain explanations of the system's actions: for instance, what policy rule applied to allow disclosure of the user's location, or why a request was denied.

PeopleFinder has been deployed in numerous field studies involving over 100 total users. Detailed log data collected during these deployments form the basis for our validation experiments discussed in the coming sections. These data also illustrate why PeopleFinder is a compelling example on which to validate our framework: first and foremost, users of PeopleFinder have demonstrated the usual difficulty specifying their privacy policies by hand, achieving an average initial accuracy of 60% [10], and are thus prime candidates for a support system such as ours.

Furthermore, the observed user behavior indicates that, when attempting to improve their policies by hand, users generally make small, incremental changes. As we will show, the space of neighboring policies is vast; we therefore conjecture that our approach, which can sample a larger subspace of neighbors than the user could realistically hope to, will assist users in selecting the best incremental change.

Contributions.

We introduce a formal model of *user-controllable policy learning* for domains where policies can be modeled as collections of condition/action rules. This includes an abstract policy transformation model along with a policy scoring function intended to capture model accuracy, complexity, and deviation from a current policy. A neighborhood search instantiation of this model has been developed and validated in the context of data obtained from deployments of our PeopleFinder application, introduced above. Simulations of our incremental policy improvement techniques on a large number of scenarios representative of the policies and audit data observed in our deployments suggest that, with our technique, users starting from the initial 60% average accuracy should expect to reach 80% accuracy within about 2 weeks of use and as much as 90% within 5 weeks. These results appear encouraging, given that users in our deployments were observed to plateau at about

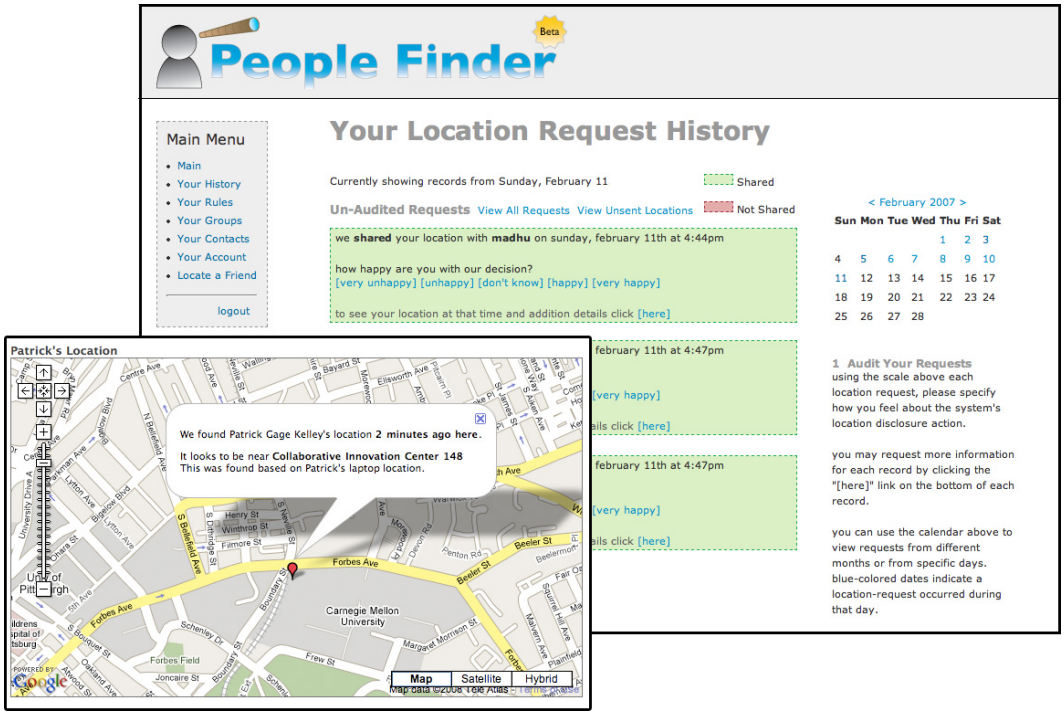

Figure 2: PeopleFinder System Feedback Interface and Location

79% accuracy when allowed to manually refine their policies over equivalent periods of time.

2. RELATED WORK

Systems applying machine learning techniques to user preferences—most often for the recommendation systems of content and products—have enjoyed great success both in industry, where the Amazon.com service is arguably the best known instance [4], and in the scientific literature, where [8, 9, 11] are representative examples. This field is surveyed in [1]. Many of these systems are, like ours, based on explicit user feedback, for example in the form of content ratings. Stumpf et al. [12] examine this feedback interaction paradigm in detail, concluding that users find it to be a highly usable manner in which to interact with systems.

Within this preference learning literature, our work is most closely related to that in the example-critiquing field. In example-critiquing interactions (c.f. [2]) users refine their preferences by critiquing example options from among a large option space. In [13], the authors extend example-critiquing with suggestions, additional candidate options drawn from among a diverse set in such a way as to glean information about the user's preferences that may not be reflected in the system's current preference model.

All of these approaches, however, restrict themselves to making black box recommendations; the user generally need not be aware of the underlying model upon which the system is basing its decisions[1]. In previous work, we also showed that black box machine learning techniques can improve upon the accuracy of user-defined privacy policies [10]. In some systems, black box techniques are augmented with explanations, and individual recommendations come with a justification [3]. In contrast, in the user-controlled policy learning approach presented in this paper, the user and the recommendation system operate hand-in-hand, improving on a common policy model, with the user free to accept, modify, or reject any recommendations before they are applied. This departs from previous approaches like that of [7], where feedback is used simply as training data for an initial learning phase.

[1]Exceptions are limited to simple models such as the use of black lists and white lists in spam filters.

3. A FORMAL MODEL OF INCREMENTAL POLICY REFINEMENT

We adopt a simple formal model of a policy, summarized in the top part of Fig. 3, that views policies as condition/action rules that specify what actions should be taken under what circumstances. We assume that some set of actions Action can be restricted according to various criteria (e.g. the time of day or identities of users generating events), all of which are captured in the set Restriction. A *rule* describes the restrictions under which a given set of actions may be executed. These restrictions are defined as a logical conjunction: all must be fulfilled before the given actions may be taken. A rule is thus defined as follows: $\mathsf{Rule} = \mathbb{P}(\mathsf{Restriction}) \times \mathbb{P}(\mathsf{Action})$. Policies themselves are then represented by a set of such rules, connected by logical disjunction; thus, a policy is simply a set of condition/action pairings, and we have $\mathsf{Policy} = \mathbb{P}(\mathsf{Rule})$. We assume that policies are consistent, in the sense that no rule or set of rules with non-disjoint premises entail actions that conflict with one another.

The set of events is represented by the set Event. Events are evaluated by the function Evaluate, which compares an event to the appropriate policy and executes either the actions

Abstract policy model	
Restriction	Set of possible restrictions
Action $\ni \perp$	Possible actions, including the null action $\perp$
Rule $= \mathbb{P}(\text{Restriction}) \times \mathbb{P}(\text{Action})$	Set of rules
Policy $= \mathbb{P}(\text{Rule})$	Set of policies
Event	Set of possible system events
Evaluate :: Policy $\times$ Event $\rightarrow \mathbb{P}(\text{Action})$	Policy enforcement decision function
Feedback	Possible user feedback (generally, Feedback $= \mathbb{N}$)
Audit :: Event $\times \mathbb{P}(\text{Action}) \rightarrow$ Feedback	Audit function
Instantiation for our friend finder system	
Weekdays	Set of weekdays
TimeSpan $=$ [StartTime..EndTime]	Time spans
WeeklyPattern $= \mathbb{P}(\text{Weekdays}) \times$ TimeSpan	Restrictions based on weekly patterns of time intervals
User	Set of users of the system
Group $= \mathbb{P}(\text{User})$	Groups of users
Restriction $=$ WeeklyPattern $\cup$ User $\cup$ Group	Set of restrictions for PeopleFinder
Action $= \{\text{Disclose}, \text{Withhold}\}$	Set of possible disclosure decisions
Feedback $= \{0, 1\}$	Set of possible feedback responses

Figure 3: Policies as Condition/Action Rules: Abstract Model and Instantiation

specified within the rule or does nothing, modeled via the null action $\perp$.

As described, we assume that the users have some means of giving feedback on decisions that the system has taken on his or her behalf. The possible feedback options are modeled via the set Feedback, which could, for instance, be a binary yes or no or a numeric scale indicating user satisfaction. It is straightforward to generalize this model and support feedback in the form of an alternate sets of actions that the user would have preferred to the actions taken by the system. This audit data, for a given event R and the associated evaluation decision D, is captured in our model via the function Audit(R, D).

Example 1. To further illustrate our policy model, we instantiate it to an appropriate formalization of the policies of PeopleFinder, illustrated in the lower half of Fig. 3. For simplicity, we ignore restrictions based on locations, but extending this model to incorporate such restrictions is straightforward. The restrictions incorporated in these policies take three forms: either a WeeklyPattern, which describes a time span valid on a set of weekdays, an individual user from set User representing a friend in the user's social network, or a group of users from set Group. In our system, the possible actions are disclosure of a user location or rejection of a request, Action $= \{\text{Disclose}, \text{Withhold}\}$, though we note that disclosure decisions need not always be binary. Returning an obfuscated version of the data requested is a strategy often employed in privacy-sensitive applications, thus one possible disclosure decision could be to return a downgraded version of the data requested. In PeopleFinder, the options for user feedback are also binary, Feedback $= \{0, 1\}$. In accordance with our convention, described below, that policies with minimal scores are most desirable, we let 0 indicates user satisfaction, and 1 user dissatisfaction. This refinement of the given model suffices to describe the policies within PeopleFinder.

Policy Transformation.

A central feature of our approach is that usability is achieved partly by ensuring that suggested changes to a user's policy are incremental, selected among a space or neighborhood of transformed policies that are close to the user's existing policy. Here, we define a framework—summarized in Fig. 4—for the description of that neighborhood of related policies.

Restrictions and actions, and by extension rules and policies, can be transformed in various ways, according to the specifics of the given system. A restriction that limits disclosure to a set of specified users, for instance, can be transformed via the deletion of one of the users or the inclusion of a new one. Our model assumes a function on restrictions, Transform, which returns the set of neighboring restrictions reachable from a given restriction by a single, incremental transition step, however that might be defined in a refinement of the model. Similarly, the GenActions function yields all incremental transformations of an action. We extend these transformations to rules via function GenRules, which, in addition to taking the union of all possible transformations on the restrictions and/or actions within the rule, can eliminate one of those restrictions or actions or add an arbitrary new one. This transformation is further lifted to policies, as reflected in function Neighbor, which considers all possible rule transformations yielded by the GenRules function, and additionally allows for the deletion of an entire rule from the policy or the addition of a new rule with no restrictions or actions, modeled as the addition of the empty rule $(\emptyset, \emptyset)$ to the policy. We illustrate an instantiation of this transition model appropriate for our friend finder in the next section.

Policy Evaluation.

We now define an objective function by which to evaluate each neighbor in order to select those that offer the greatest policy improvement. Here, we assume that the objective is

$$\begin{array}{ll}
\mathsf{Transform} :: \mathsf{Restriction} \rightarrow \mathbb{P}(\mathsf{Restriction}) & \text{Restriction transformation function} \\
\mathsf{GenActions} :: \mathsf{Action} \rightarrow \mathbb{P}(\mathsf{Action}) & \text{Action transformation function} \\
\mathsf{GenRules} :: \mathsf{Rule} \rightarrow \mathbb{P}(\mathsf{Rule}) & \text{Rule generation function, where} \\
\quad \mathsf{GenRules}((R,A)) = & \bigcup_{r\in R} \bigcup_{r'\in \mathsf{Transform}(r)} (r', A) \cup \\
& \bigcup_{a\in A} \bigcup_{a'\in \mathsf{GenActions}(a)} (R, a') \cup \\
& \bigcup_{r\in R} R \setminus \{r\} \cup \bigcup_{r\in \mathsf{Restriction}} R \cup \{r\} \\
& \bigcup_{a\in A} A \setminus \{a\} \cup \bigcup_{a\in \mathsf{Action}} A \cup \{a\} \\
\mathsf{Neighbor} :: \mathsf{Policy} \rightarrow \mathbb{P}(\mathsf{Policy}) & \text{Neighbor generation function, where} \\
\quad \mathsf{Neighbor}(P) = & \bigcup_{p\in P} \mathsf{GenRules}(p) \\
& \bigcup_{r\in P} P \setminus \{r\} \cup \\
& P \cup \{(\emptyset, \emptyset)\}
\end{array}$$

Figure 4: Abstract Policy Transformation Model

to minimize the value of a function E on a policy P given a history of events R that also incorporates user feedback.

We formulate objective functions with respect to three factors. The most obvious criterion on which to score policies is the amount of negative feedback they generate. Assuming a numeric scale of user feedback, where higher numeric values indicate lower satisfaction (e.g. 0 indicates highest user satisfaction) this amounts to the sum of $\mathsf{Audit}(r, \mathsf{Evaluate}(P, r))$ for all events $r \in R$.

Maximizing user satisfaction, however, does not protect against overfitting, nor is it a guarantee of understandability. Our second factor penalizes fine-granularity for the twofold purpose of preventing overfitting and giving precedence to less complex policies that are likely to be more readable. We assume a function $\mathsf{Complex}(P)$ that assigns a measure of complexity to policy P, where 0 is the least complex. At this level of abstraction, the most obvious complexity criteria are the number of rules and the number of restrictions and actions per rule.

Finally, given that policy suggestions should be user-comprehensible, we also penalize suggestions that result in greatest deviation from the user's existing policy. We assume a function that assigns a distance metric to two policies P and Q, $\Delta(P, Q)$, where $\Delta(P, P) = 0$. Usability studies will follow to research how much deviation users find acceptable, but observed usage of our friend finder application indicates that most users update their policies by small, incremental modifications. In general, $\Delta(P, Q)$ can be difficult to compute for arbitrary P and Q. In practice, however, one can often get around this problem: in the implementation discussed in the next section, for instance, we view the space of neighbors of a policy P as a graph and define $\Delta(P, Q)$ simply as the number of edges (i.e. incremental policy transitions) required to reach Q from P.

Assuming penalty coefficients of γ, ρ, and φ for user satisfaction, complexity, and deviation, respectively, and letting P' be the user's current policy and R be a history of events, we can define a general evaluation function for policies in this abstract model as follows: $E(P' \hookrightarrow P, R) = \gamma \sum_{r\in R} \mathsf{Audit}(r, \mathsf{Evaluate}(P, r)) + \rho\mathsf{Complex}(P) + \varphi\Delta(P, P')$.

This policy evaluation framework, like our model itself, can be refined as appropriate for specific applications and settings. In the study we discuss in the next section, we evaluate PeopleFinder policies based on the number of dissatisfied (i.e. 1-valued) audits they generate, and complexity as defined by the number of rules contained in a policy, and the number of groups, users, and weekly patterns contained within those rules. Policy deviation is not penalized explicitly. Rather, we preclude deviation above a certain threshold by restricting the number of transition steps allowed when generating the space of neighbors.

4. RESULTS

In this section, we present an instantiation of our model for incremental policy refinement in the context of the PeopleFinder application introduced above. We proceed to describe a simple neighborhood search implementation of our user-controllable policy learning approach developed for this application. We report on its performance on simulated scenarios based on data derived from experimental deployments of the PeopleFinder system and compare it with the performance achieved by users who manually modified their policies during the course of these deployments.

Policies.

Fig. 5 shows a graphical representation of a simple PeopleFinder policy, as introduced in Fig. 3, with ten contacts organized into four groups (with some overlap, specifically Cindy and Heath). Black bars indicate times when disclosure is permitted—to the groups named on the left, on the weekdays listed on the right. Only the group "Family" has access to our user's location all the time, every day.

A Neighborhood Search Implementation.

A simple, yet elegant implementation of our user-controllable policy learning approach involves using neighborhood search to explore incremental modifications of the user's current policy. These modifications can be generated using transformation operators that are selected to cover a meaningful set of easy-to-understand modifications to the user's current policy. Results presented in this paper used the following set of policy transformation operators:

- The deletion of an existing rule, or the addition of a new rule permitting disclosure to a given user during a given time span on a given day.
- The expansion or contraction of either the start or end of a time-span by up to an hour.
- The deletion of a day from a duration within a rule, or the addition of a day to a rule duration.
- The addition of a person to a group, or the deletion of a person from a group.

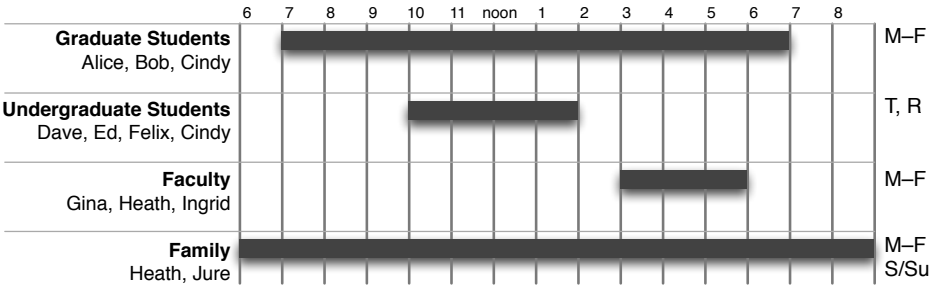

Figure 5: A simple PeopleFinder policy

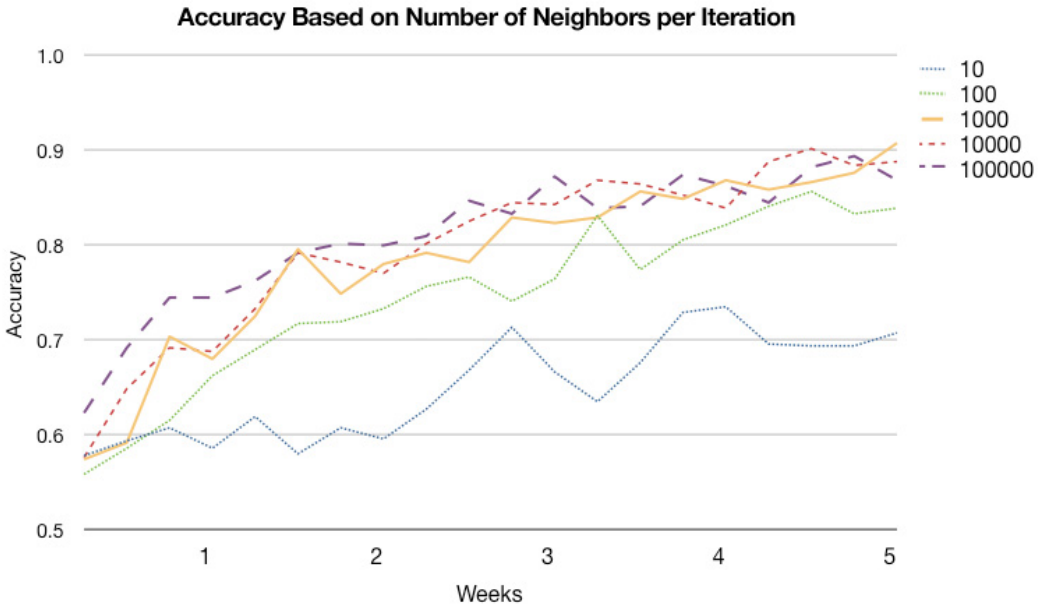

Figure 6: While both ten and one hundred neighbors do not show rises in accuracy, one thousand, ten thousand, and one hundred thousand neighbors generated per iteration all show similar returns in accuracy, about thirty percent over four weeks.

In the results reported here, suggestions were selected by randomly generating and evaluating a large number of neighbors of the user's current policy. Each neighbor was equally likely to be selected—clearly, more sophisticated implementations could assign different probabilities to different operators. We also experimented with instantiations of this search procedure that varied based on the number of neighbors generated for a given policy as well as the number of successive moves (or policy transformations) allowed at each step. Intuitively, several successive moves allow the procedure to explore a wider, more diverse neighborhood, though at the risk of suggesting policy modifications that are more difficult for the user to understand. It was therefore important to evaluate the sensitivity of our approach to variations in these parameter values and see to what extent a very limited number of moves (e.g. just one policy transformation) might be sufficient to generate suggestions that would yield meaningful improvements in accuracy.

Each time the user-controllable learning procedure is invoked, it generates and evaluates a number of policy modifications (based on accuracy, complexity and deviation from the current policy) and uses the top rated transformations to suggest possible policy modifications to the user. In the experiments reported in this paper, we limited ourselves to generating a single suggestion each time, namely the policy transformation with the highest score among all those generated by the neighborhood search procedure.

Empirical Setup.

To validate our approach, we used data derived from experimental campus deployments of our PeopleFinder application. These deployments, which spanned between 1 and 8 weeks, involved a total of over 100 participants. The pilots, which confirmed that users often have great difficulty articulating their policies, also provided a baseline against which we were able to compare the performance of our user-controllable policy learning algorithms. Specifically, detailed logs collected during the deployments of PeopleFinder were used to characterize the complexity of initial policies defined by users, the average number of daily requests users received,

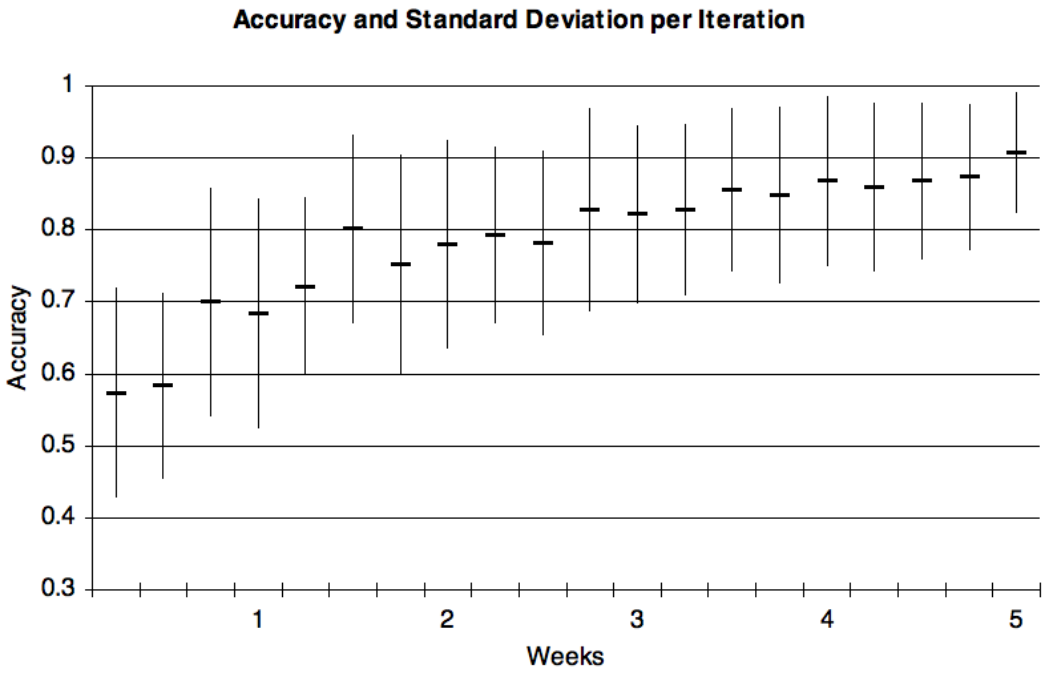

Figure 7: Average accuracy and standard deviation

and the frequency with which users revise these policies. This information was, in turn, used (1) to simulate a large number of scenarios representative of user behavior observed during our deployments and (2) to extrapolate plausible usage scenarios for our user-controllable policy learning procedure. In the experiments reported in this section, we present results from a first set of these scenarios, in which users received an average of 5 location requests per day, audited and revised their policies every other day, with policy revision limited to a single policy modification. Assuming this level of usage, we report our results on policy improvement based on the number of weeks using the system. We further assume the modification selected by the user was the top ranked suggestion generated by the neighborhood search implementation of our policy learning algorithm, using all previously audited data.

Results.

Fig. 6 shows the sensitivity of our policy learning algorithm to the number of neighbors generated. As can be seen, a value of 1,000 neighbors was generally sufficient to provide for fairly rapid improvements in accuracy and greater values did not significantly increase performance.

Fig. 7 shows average accuracy with standard deviations across fifty runs of our simulation. It can be seen that while, early on, the system is only predicting on average about six incoming decisions correctly out of ten (which is consistent with the accuracy of policies defined by users in actual deployments of PeopleFinder), after about 2 weeks, accuracy reaches about 80% and even climbs to about 90% if given an extra 2 to 3 weeks. In comparison, the accuracy of policies refined manually by PeopleFinder users plateaued at around 79%. While a strict comparison between these two numbers would be inappropriate, these results suggest that incremental policy learning of the type advocated in this paper can likely help users define more accurate policies.

Experiments in which the neighborhood search procedure was allowed to explore multiple consecutive policy transformations provided for slightly faster convergence, though at the expense of higher computational overhead and with the drawback of generating suggestions that are likely more difficult for users to understand.

In summary, even with the large number of parameters we have controlled above, we have shown that with small numbers of audited requests per week—within the range of our earlier field studies—we can generate suggestions that, if taken, would significantly increase policy accuracy.

Most of all, we see in these results similar and occasionally even better accuracy than we have seen in our earlier field studies where users manually edited their policies over time. This means that making policies change only in user-understandable increments based on our neighborhood search should help users bring their policies to convergence more quickly than without aid.

5. CONCLUDING REMARKS

Users increasingly expect to be able to customize applications and systems they interact with. This includes the ability to customize a variety of policies that users are generally not good at accurately defining. Machine learning techniques have been used to overcome this challenge by refining default or user-defined policies by leveraging user feedback. So far, these techniques have generally been configured as black box solutions that severely restrict the ability of users to manipulate the models learned by the system. In contrast, in this paper, we have introduced a user-controllable policy learning approach that enables the user and the system to work hand in hand on refining policies by working on a common model. Our work accomplishes

this by learning incremental policy modifications which are presented to the user as suggested changes to their current system policy.

In this article, we introduced a formal model of incremental policy modifications and described a neighborhood search implementation of it. An important contribution of our work has been to show that incremental policy modifications seem sufficient to yield meaningful improvements in policy accuracy over relatively short periods of time. This result is important, as it suggests that it is possible to reap the benefits of machine learning techniques while moving away from their traditional black box configurations, all the while leaving the user in control. User retention of control reduces the risk of the system introducing particularly bad policy modifications. This is of particular importance in the security and privacy spheres but is desirable in all settings where the lack of control offered by traditional machine learning configurations is a detriment to the user experience.

6. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[2] Boi Faltings, Pearl Pu, Marc Torrens, and Paolo Viappiani. Designing example-critiquing interaction. In *Proceedings of the 2004 International Conference on Intelligent User Interfaces*, User modeling I, pages 22–29, 2004.

[3] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 241–250, New York, NY, USA, 2000. ACM.

[4] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[5] Roy A. Maxion and Robert W. Reeder. Improving user-interface dependability through mitigation of human error. *International Journal of Man-Machine Studies*, 63(1-2):25–50, 2005.

[6] Christena Nippert-Eng. Privacy in the United States: Some implications for design. *International Journal of Design*, [Online] 1:2, Aug 2007. Available at `http://www.ijdesign.org/ojs/index.php/IJDesign/article/view/67/30`.

[7] Michael J. Pazzani. Representation of electronic mail filtering profiles: a user study. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces*, pages 202–206, New York, NY, USA, 2000. ACM.

[8] Michael J. Pazzani and Daniel Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331, 1997.

[9] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, and John Riedl. Getting to know you: learning new user preferences in recommender systems. In *IUI*, pages 127–134, 2002.

[10] Norman Sadeh, Jason Hong, Lorrie Cranor, Ian Fette, Patrick Kelley, Madhu Prabaker, and Jinghai Rao. Understanding and capturing peopleâĂŹs privacy policies in a people finder application. In *Proceedings of the 5th International Workshop on Privacy in UbiComp (UbiPriv'07)*, September 2007.

[11] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Min. Knowl. Discov.*, 5(1/2):115–153, 2001.

[12] Simone Stumpf, Vidya Rajaram, Lida Li, Margaret Burnett, Thomas Dietterich, Erin Sullivan, Russell Drummond, and Jonathan Herlocker. Toward harnessing user feedback for machine learning. In *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces*, pages 82–91, New York, NY, USA, 2007. ACM.

[13] Paolo Viappiani, Boi Faltings, and Pearl Pu. Preference-based search using example-critiquing with suggestions. *J. Artif. Intell. Res. (JAIR)*, 27:465–503, 2006.