Designing for Effective End-User Interaction with Machine Learning

Saleema Amershi

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington
2012

Reading Committee:
James A. Fogarty, Chair
Daniel S. Weld
Desney S. Tan

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Designing for Effective End-User Interaction with Machine Learning

Saleema Amershi

Chair of the Supervisory Committee:
Associate Professor James A. Fogarty
Computer Science & Engineering

End-user interactive machine learning is a promising tool for enhancing human capabilities with data. Recent work has shown that we can create specific applications that employ end-user interactive machine learning. However, we still lack a generalized understanding of *how to design effective end-user interaction with machine learning*. This dissertation advances our understanding of this problem by demonstrating effective end-user interaction with machine learning in a variety of new situations and by characterizing the design factors affecting the end-user interactive machine learning process itself. Specifically, this dissertation presents (1) new interaction techniques for end-user creation of image classifiers in an existing end-user interactive machine learning system called CueFlik, (2) a novel system called ReGroup that employs end-user interactive machine learning for the purpose of access control in social networks, (3) a novel system called CueT that supports end-user driven machine learning for computer network alarm triage, and (4) a novel design space characterizing the goals and constraints impacting the end-user interactive machine learning process itself. Together, these contributions can move us beyond ad-hoc designs for specific applications and provide a foundation for future researchers and developers of end-user interactive machine learning systems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to begin by thanking my wonderful advisor James Fogarty for his guidance and endless support throughout my graduate career. James has been my greatest champion, never giving up on me and never letting me give up on myself. I will no doubt continue to seek his advice long into the future. I also want to thank my bonus advisor, Desney Tan, for his unwavering encouragement. His insight and drive continues to inspire me. This dissertation would not have been possible without these two amazing people. I feel fortunate to have had the opportunity to work and know them.

I would also like to thank the many extraordinary people I have had the privilege of working with. In particular, I would like to thank Merrie Morris for taking a chance on me; Dan Weld for his wisdom and thought-provoking insight; Ashish Kapoor for his positivity and nonjudgmental explanations; Bongshin Lee for her infinite kindness; and Ed Chi for his energy and enthusiasm. A special thanks also goes to James Landay for believing in me; Jacob Wobbrock for always making time for me; and Lindsay Michimoto for her support and encouragement.

Many thanks also go to my colleagues and friends in my extended research group at the University of Washington and beyond. I especially want to thank Morgan Dixon, Scott Saponas, Jon Froehlich, Michael Toomim, Mira Dontcheva, Shiri Azenkot, Lydia Chilton, Katie Kuksenok, Travis Kriplean, Kate Everitt, Hao Lu, Leah Findlater, Rebecca Fiebrink, Raphael Hoffman, Jay Chen, Kevin Li, Parmit Chilana, and Tejinder Judge.

I am also grateful to my awesome housemates and friends for the countless memories. Thanks to Nodira Khoussainova for being a constant source of support and friendship. Thanks to Alex Jaffe and Kayur Patel for making me laugh until my cheeks hurt. Thanks also to Ran Tao for the endless supply of excitement, Liz Tseng for all the yummy food, and Dan Halperin and Ian Simon for the philosophizing.

I could not have come this far without the encouragement and support of my family. Thanks especially to my incredible Mom for her limitless love and silliness. Thanks to my Dad for inspiring me. Thanks also to my new family, Daryl, Sherri, Craig, and Cody McDonald for openheartedly welcoming me into their lives. Thanks also to my de facto family Sukhpreet Dhillon, Dal Dhillon, Rekha Brar, Nancy Parmar, and the girls, for always being there when I needed them the most. I owe you girls some robots!

Finally, I could not have done this without the tireless support of my amazing husband and best friend Ian McDonald. He inspires me to be a better person every day.

This work was supported in part by the National Science Foundation under awards IIS-0812590, IIS-1016713, and OAI-1028195, by Office of Naval Research award N00014-06-1-0147, by the WRF / TJ Cable Professorship, by a Google Faculty Research Award, by a Microsoft Research Award, by three internships at Microsoft Research, by an internship at Google Research, by an internship at IBM Research, by the Kumar and Roberta L. Bhasin Endowed Fellowship, by the Google Anita Borg Scholarship, by the Microsoft Endowed University Fellowship, and by the author's wonderful colleagues, family, and friends.

DEDICATION

To my mom, who was always taking me to the Science Museum.

Chapter 1
Introduction

1.1 Motivation

It is widely believed that powerful information is locked within the vast amounts of data we now have access to and harnessing this information can enhance our capabilities and improve our lives (e.g., Anderson and Rainie, 2012; World Economic Forum Report, 2012). For example, a recent report by the Pew Internet & American Life Project (Anderson and Rainie, 2012) stated that the majority of technology experts surveyed believe that "big data" will have significant effects on human productivity and decision making. Specifically, many experts believe that data will have a broad impact on our economy, national security, health care, education, and green technology industries. Accordingly, many government agencies, universities, and large corporations have started investing in big data initiatives dedicated to developing new ways of tapping into this transformative resource (e.g., Weiss and Zgorski, 2012; Denison, 2012; Carstensen, 2012). For example, the federal government recently unveiled a $200 million research and development initiative for advancing the state of the art in collecting, storing, analyzing, and generating new knowledge from big data (Weiss and Zgorski, 2012).

A key approach for transforming raw data into usable information is *machine learning*. Machine learning is the process of teaching computers to automatically recognize concepts or patterns of interest in data. For example, machine learning has been used to convert everyday search queries into models that can track the spread of influenza (Ginsberg *et al*, 2009). In the same way, machine learning has been used over incident reports to predict violent outbreaks in war torn regions (Zammit-Mangion *et al,* 2012; Beckhusen, 2012), over MRI scans for early detection of Alzheimer's disease (Kloppel *et al*, 2008), over social network streams to understand and monitor public sentiment of political parties and issues (Hardy, 2012), and over observed human behaviors to aid financial decision making (Duhigg, 2012; Cha, 2012).

Unfortunately, the complexity of machine learning has largely restricted its practice and use to experts and skilled developers. A consequence of this is that everyday people are then inherently limited to using data in developer-conceived ways. For example, our applications can only detect or monitor concepts that a developer has regarded as interesting or meaningful. In addition, because experts are a scare resource, relying on them for machine learning severely limits the number of concepts or patterns we can ever extract from data (Rooney, 2012; Anderson and Rainie, 2012).

Despite these limitations of relying on experts to make use of data, most large-scale big data initiatives continue to focus on developing new tools and technologies for expert use (e.g., Weiss and Zgorski, 2012; Denison, 2012; Carstensen, 2012). However, we assert that regardless of the sophistication of future tools and technologies, experts cannot possibly foresee the countless variety of concepts people might be interested in nor can they drive machine learning to extract all of these concepts from our data. Therefore, to take full advantage of the opportunity that big data provides and realize a vision of enhanced human productivity and capabilities from data, we must equip everyday people with the ability to drive machine learning themselves.

1.2 Objectives and Hypothesis

This dissertation examines the fundamental process by which end-users can interact with machine learning systems, known as end-user interactive machine learning (see Figure 1.1). In the end-user interactive machine learning process, a person iteratively guides a machine towards learning a desired concept and inspects feedback illustrating the machine's current understanding.

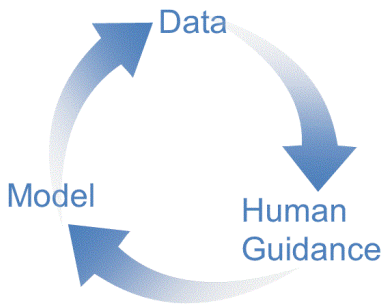

Figure 1.1: The cycle above illustrates the fundamental aspects of the end-user interactive machine learning process. In this process, a person iteratively guides a machine towards learning a desired concept. The person then inspects the system's current understanding and decides how to proceed with further guidance. The learned concept is encoded by the machine as a model that can configure automated behaviors on data.

Recent work has demonstrated several applications of end-user interactive machine learning in a variety of compelling application domains. For example, Fails and Olsen's Crayons system supports end-user interactive machine learning for image segmentation (2003). Ritter and Basu also demonstrated the potential for end-user interactive machine learning in complex file selection tasks (2009). Dey *et al*'s aCAPpella system demonstrated end-user interactive machine learning for end-user driven context-aware computing (2004). In addition, it is becoming increasingly common for online retailers (e.g., Amazon.com, eBay.com) and service providers (e.g., Pandora.com, Netflix.com) to employ interactive machine learning for recommending items to purchase or view.

These applications provide evidence of the utility of interactive machine learning. However, we still lack a generalized understanding of *how to design effective end-user interaction with machine learning*. This is evidenced by the fact that most real-world applications of end-user interactive machine learning still design interaction in much the same way. For example, most commercial recommender systems support end-user interaction in one of two ways, by enabling indiscriminate item ratings (e.g., 'likes' and 'dislikes' on items) or by implicitly obtaining item feedback from observations of user behavior. While this approach simplifies interaction, it can also create a frustrating user experience if a person cannot effectively guide the system (Rao et al, 2009; Zaslow, 2002). Moreover, this approach may be inappropriate for interactive machine learning in different scenarios. For example, when an end-user's primary goal in interacting with a machine learning system is to create an accurate and reusable model (as is the case with many of the recent end-user interactive machine learning systems presented by the research community), they may desire more feedback or guidance about how to effectively steer the system.

At the other extreme, the traditional active learning approach to designing end-user interaction with machine learning is to force a person to label items that provide the machine with the greatest information gain. While this favors efficient learning, it can be equally frustrating to an end-user by discounting their abilities and treating them like a passive oracle of information (Baum and Lang, 1992; Gurevich *et al*, 2006).

In the research community, much of the focus on improving the design of end-user interaction with machine learning-based systems has been on attempting to increase the transparency and understandability of these inherently complex systems (e.g., Herlocker *et al*, 2000; Tintarev and Masthoff, 2007; Lim and Dey, 2009; Kulesza *et al*, 2011). For example, some researchers have advocated the use of detailed explanations of system behavior to increase understandability and in turn improve end-user trust in machine learning (e.g., Lim and Dey, 2009; Kulesza *et al*, 2011). However, explanations might impede

efficiency during interaction. In addition, explanations may be unnecessary when output of the machine learning will be regulated by the end-user (as opposed to when the system will be taking autonomous actions on behalf of the user) or when the system is operating in a low risk environment (e.g., automatically selecting songs to play).

The goal of this dissertation is to advance our understanding of how to design effective end-user interaction with machine learning. To this end, I argue that what is needed for a general understanding of how to design end-user interaction with machine learning is a clearer understanding of (1) the variety of situations in which end-user interactive machine learning may be used and (2) the factors that can impact and constrain an end-user's interaction with machine learning in those situations. Therefore, it is my thesis that:

Interactive machine learning can give end-users new leverage on data in a variety of applications, and we can move beyond an ad-hoc understanding of how to design such applications by characterizing the goals and contexts surrounding the interactive machine learning process itself.

1.3 Summary of Contributions

This dissertation makes the following contributions towards advancing our understanding of how to design effective end-user interaction with machine learning:

- A novel design space characterizing the factors that should be considered in designing end-user interaction with machine learning (Chapter 3). This design space also outlines dimensions that have been explored in the design of existing systems and research. Together these design factors and dimensions can help us navigate the space of potential solutions for designing end-user driven machine learning, thereby providing a foundation and reference for future researchers and developers. In addition, this design space can facilitate communication across existing disciplines that employ machine learning for domain specific purposes.

- The design and evaluation of new interaction techniques for facilitating end-user creation of image classifiers in an existing interactive machine learning system called CueFlik (Chapter 4). In particular, here we examine the design of three important aspects of the interactive machine learning process: (1) illustrating the system's current understanding of a concept, (2) guiding end-users to select effective training examples, and (3) enabling end-user exploration of multiple potential models.

- The design and evaluation of a novel system called ReGroup that employs end-user interactive machine learning for the purpose of access control in social networks (Chapter 5). In designing

ReGroup's interactive machine learning, we specifically explored new interaction techniques for: (1) supporting interactive machine learning while preserving end-user control, (2) indirectly training an effective classifier, (3) minimizing frustration caused by unlearnable groups, and (4) integrating human knowledge through interaction with both examples and features.

- The design and evaluation of a novel system called CueT that supports end-user driven machine learning for the high-impact process of computer network alarm triage (Chapter 6). Here we specifically investigate new techniques for (1) learning over a stream of data where the classes of interest are unknown *a priori* and evolve constantly and (2) visualizing model quality to reduce errors.

1.4 Scope

There are many ways in which end-users can interact with machine learning-based systems. These range from interaction with fixed or predefined models created by experts to interaction with the purpose of defining new models or concepts of interest. During end-user interaction with fixed models, an end-user might feed data to the model and use predicted outcomes, but their interaction will not change the behavior of the system. An example of this is an application that can recognize pre-defined behaviors such as walking or running, but cannot adapt to the unique behaviors of individuals. In contrast, when defining new concepts, there is typically some sort of loop over the model, outcome, or data involved in the interactive machine learning process. For example, when the loop is over the model, the learned model itself is changing as a result of end-user guidance. Similarly, a loop over the data might involve the creation of new classes in the model that impact the predictions of the system (e.g., recommender systems that make predictions based on end-user specified classes of interest and but use a fixed similarity metric).

In this dissertation, we focus on the latter end of this spectrum. That is, we focus on interaction involving an explicit end-user interactive machine learning loop. It should be noted, however, that some design techniques discussed throughout this dissertation could transfer from designing interaction with fixed models to designing interactive machine learning and vice versa. We therefore point out such cases, particularly when design solutions from interaction with fixed models can further our understanding of how to design end-user interaction with machine learning. For example, feedback techniques for displaying recommendations from recommender systems driven by fixed models (e.g., using demographic or stereotypical information) could transfer to recommender systems using models that evolve based on end-user interaction. Therefore, it is important that we learn from these systems as well as from systems involving an explicit end-user interactive machine learning loop.

In this dissertation, we also only consider people with little to no machine learning expertise interacting with a machine learning system. This includes everyday people as well as domain experts or developers untrained in the art of statistical machine learning. For example, untrained developers desiring to build machine learned models into their end-user facing applications may face difficulties creating those models without appropriate guidance or support.

1.5 Outline

The rest of this dissertation is organized as follows:

Chapter 2 reviews related work involving end-users interacting with machine learning. This includes overviewing related fields that employ interactive machine learning as a tool to help end-users accomplish domain specific tasks (e.g., information retrieval, recommender systems, intelligent user interfaces). I also discuss related work in understanding how we can or should design end-user interaction with machine learning based systems.

Chapter 3 presents a novel design space for end-user interaction with machine learning systems. This design space is informed by the same body of related work reviewed in Chapter 2 as well as my own experiences designing end-user interactive machine learning systems. In this design space, I characterize factors impacting how the interaction between an end-user and a machine learning system should be designed. These factors include potential end-user goals and constraints on the interaction. I also identify and describe design dimensions that have been explored in previous systems to achieve various interaction goals.

In Chapter 4, I explore the design of new interaction techniques for facilitating end-user creation of image classifiers in an existing interactive machine learning system called CueFlik. Specifically, I present new overview-based feedback techniques for both illustrating CueFlik's current understanding of a desired visual concept and for guiding end-users to select effective training examples during interaction. My evaluation shows that overview-based feedback leads end-users to create better models with CueFlik than the best performing feedback technique in prior research. In Chapter 4, I also explore new revision and comparison mechanisms for enabling end-user exploration of multiple potential models during interaction with CueFlik's interactive machine learning. A second evaluation shows that revision enables people to create better models with CueFlik than when revision is absent and revision is consistent with how people expect to interact with their applications.

Chapter 5 presents ReGroup, a novel system I developed that employs end-user interactive machine learning to help people create custom groups on-demand in online social networks. In particular, ReGroup

uses interactive machine learning to recommend additional group members to end-users as they create custom groups in their social network. In designing interaction with ReGroup, I explore new techniques for supporting interactive machine learning while preserving end-user control, indirectly training an effective classifier for group member recommendation, and integrating human knowledge through interaction with both examples and features. My evaluation of ReGroup compared to the traditional method of creating custom on-demand groups in social networks (i.e., search-by-name and scrolling through an alphabetical list) shows that each method effectively supports a different class of groups. The traditional method is efficient for small well-defined groups, whereas ReGroup is more effective for larger and more varied groups.

In Chapter 6, I present CueT, a novel system I developed to support end-user interactive machine learning for the high-impact and dynamic process of computer network alarm triage. In this chapter, I present new techniques for interactive machine learning over a stream of data where the classes of interest are not known *a priori* and evolve constantly. I also present a novel visualization for depicting model quality during the interactive machine learning process. My evaluation of CueT compared to the state-of-the-art approach demonstrates that CueT increases both speed and accuracy of computer network alarm triage.

Finally, Chapter 7 concludes with a discussion of the research presented in this dissertation and suggests opportunities for future research. In this chapter, I also propose guidelines for the design of future interactive machine learning systems.

Chapter 2
Related Work

In this chapter, we review related work involving end-users interacting with machine learning. This chapter is divided into two sections:

- In Section 2.1, we overview related fields that sometimes employ interactive machine learning as a tool to help end-users accomplish domain specific tasks (e.g., information retrieval, recommender systems, intelligent user interfaces). We also discuss recent systems that have started to push the boundaries of what end-users can use machine learning for and that motivated our own research in this space. For each of these fields, we discuss where interactive machine learning plays a role and characterize the typical interactive loop applied. Where appropriate for context, we also discuss aspects of each field that we do not consider instances of end-user interactive machine learning.

- In Section 2.2, we discuss related work in understanding how we can or should design end-user interaction with machine learning based systems. Much of this existing work concerns designing end-user interaction with machine learning in specific domains or within more complex intelligent systems that may embed machine learning as a component of a larger application. However, many of the issues raised and solutions proposed in these related works apply to the design of end-user interaction with machine learning.

Throughout this chapter, we discuss related work from a human interaction perspective. We do not discuss specific machine learning algorithms used because often, as in our own experiences, several different algorithms may be suitable for a given interaction or task goal. The choice of algorithm to use is then typically the result of examining tradeoffs between desired interaction techniques for achieving those goals (see Chapter 3 for an examination of potential interaction goals and techniques for achieving them).

As discussed previously in Chapter 1, we consider something an instance of end-user interactive machine learning when there is some end-user driven interactive loop over the model, outcome, or data involved in the machine learning. In most cases, this interactive machine learning loop fundamentally impacts the way people can interact with a machine learning system. However, there are instances in which design techniques discussed in this chapter could transfer to non-interactive machine learning based systems and vice versa. For example, some fixed model or rule-based systems may operate similarly to machine learning based systems in terms of the feedback or control techniques implemented. Therefore, throughout this chapter, we note situations in which this transfer may occur.

Note that this review is not intended to be comprehensive. This review is meant to orient the reader as to the breadth of applications of end-user interactive machine learning and to highlight issues that have been raised about how to design interaction with machine learning in those applications. In Chapter 3, we present our design space for end-user interaction with machine learning which draws on the work discussed in this chapter as well as our own experiences with designing end-user interactive machine learning systems (discussed in Chapters 4 to 6). In outlining this design space, we present a closer examination of the different interaction techniques that have been used in the previous systems and research discussed in this chapter.

2.1 Interactive Machine Learning as a Tool in Related Fields

Here we discuss related fields that use interactive machine learning as a tool for accomplishing domain specific tasks (e.g., search in information retrieval, filtering in recommender systems, activity detection in context-aware computing). We also discuss areas of work often defined by their interaction paradigm but which may use interactive machine learning to achieve a variety of domain specific goals (e.g., programming by demonstration is an interaction paradigm which has been used for automating web based tasks as well as for teaching context-aware computing systems). These fields are therefore not necessarily mutually exclusive (e.g., recommender systems are often defined as a type of adaptive or intelligent system). In describing each field, we characterize the typical interactive machine learning loop and give concrete examples for illustrative purposes when necessary. We also discuss some common interaction issues studied in each area.

2.1.1 Information Retrieval

Information Retrieval is a broad field of research generally concerned with helping people access, retrieve and organize unstructured information (as opposed to structured information that can more easily be stored and retrieved from databases). Typically, this unstructured information originates from large corpora like the Web. Research in this field covers a range of topics (e.g., indexing, query optimizations,

storage and architecture), but here we discuss topics relevant to end-user interaction with machine learning. For good reviews of information retrieval see Ghorab *et al* (2012) and Greengrass (2000).

One of the major and most public facing areas of information retrieval is search, particularly with the advent of the Web search engine. In typical search, a person issues a keyword based query to communicate their information need to a machine. That query is then used to retrieve candidate documents (e.g., web pages, text or images) which are typically ranked according to importance or estimated relevance and displayed to the end-user. Retrieval and ranking is often handled by static matching algorithms (e.g., by measuring the similarity between query terms and documents in a corpus according to a static metric) or global and aggregate notions of importance or document quality (e.g., Agichtein *et al* 2006; Smyth and Balfe, 2006). Therefore, this standard search process alone typically does not involve interactive machine learning according to our definition (i.e., there is no interactive loop involving the machine learning).

When search does involve an interactive machine learning loop, it is typically used for a process known as relevance feedback (Salton and Buckley, 1990; Harman, 1992; Greengrass 2000). Relevance feedback is the process by which end-users can (iteratively) provide information to the system about the relevance of documents presented to them. The system then uses this additional information to modify the search results presented (e.g., by re-ranking or filtering the results (Stamou and Ntoulas, 2009; Teevan *et al*, 2005; Speretta and Gauch, 2005)) or to retrieve new results (e.g., through automatic query refinement or expansion and re-retrieval (Ruvini, 2003; Pitkow *et al,* 2002; Shen *et al,* 2005)).

There are two main classes of feedback techniques in search based relevance feedback: explicit and implicit. With explicit feedback, a person directly supplies relevance information to the machine (e.g., via ratings on documents (Harman, 1992)). However, the additional effort required from the end-user for this type of feedback is generally believed to be cumbersome and obtrusive (Gauch *et al*, 2007; White *et al*, 2002). This has led to much research on techniques for implicitly obtaining relevance information (e.g., based on system observed behaviors such as document clicks or dwell times (Shen *et al,* 2005; Stamou and Ntoulas, 2009; Teevan *et al*, 2005; White *et al*, 2002; White, 2005)).

While research has demonstrated that relevance feedback can improve the quality of search results, particularly when it is difficult for a person to formulate their query (e.g., image search (Crucianu *et al*, 2004)) or their information needs evolve, conventional relevance feedback is rarely used in commercial Web search engines (Manning *et al*, 2008). Many commercial search engines do, however, provide personalized search results in which retrieved documents are ranked or filtered based on historical

evidence of user interests such as previous queries or documents views (Ghorab *et al*, 2012). We do consider this type of personalized search as interactive machine learning. However, the interactive loop here is generally more prolonged than the short-lived relevance feedback loop typically guided by a person's current information need. Personalized search is therefore akin to user modeling in recommender and intelligent systems as will be discussed in subsequent sections.

Another major topic area in information retrieval that is increasingly using end-user interactive machine learning involves the process of organizing or categorizing large corpora of data. This area typically includes document classification (e.g., spam filters and email classifiers), document clustering, and annotation (e.g., tagging documents).

Document classification in information retrieval typically refers to the classification of objects from a streaming data source and is therefore sometimes called routing or filtering. One of the most studied applications of document classification of this kind involves personal email classifiers and spam filters. Most commercial email applications support hand-crafted rule-based email filters (Crawford *et al*, 2001; Pazzani, 2000). These rules and filters can be used, for example, to automatically file or tag emails that originate from a specific sender or that contain a specific word. However, because such filters are manually constructed and fixed, we do not consider them instances of interactive machine learning.

Cases involving end-user interactive machine learning include classifiers learned from explicit or implicit end-user behaviors (e.g., Segal and Kephart, 1999; Crawford *et al*, 2001; Boone, 1998; Sahami *et al*, 1998; Aberdeen *et al*, 2010; Payne and Edwards, 1997). For example, Segal and Kephart's MailCat system (1999) monitors an end-user's manual email filings and learns a classifier which can be used to make predictions for future filings. Predictions are presented in a ranked list and can be used or ignored, providing additional feedback to the learning system. Crawford *et al*'s I-EMS system (2001) similarly learns email filters based on previous actions but also includes a sliding window which discards old emails in order to try and adapt to a person's changing behaviors. Gmail's Priority Inbox (Aberdeen *et al*, 2010) also learns from previous actions to determine which emails might be important or not (based on behavioral information including interaction histories between the senders and receiver and keywords within the emails) and then automatically organizes emails into these categories. Priority Inbox also allows for explicit feedback by supporting manual labeling or correcting of the system's predictions.

Document clustering generally refers to the grouping of items from a relatively static corpus (e.g., a large but fixed corpus of papers that need to be organized by topic). Interactive document clustering involves the end-user in this process (e.g., Drucker *et al*, 2011; Hearst *et al*, 1995; Baker *et al*, 2009). For example,

Hearst *et al*'s Scatter/Gather system (1995) automatically clusters documents into topical categories. A user can then iteratively select clusters of interest (and remove irrelevant clusters) which are then re-clustered by the system for further examination. This Scatter/Gather technique can help people better understand and process large amounts of data (e.g., by categorizing results that might be retrieved from a search engine). Baker *et al*'s document clustering system (2009) also supports interactive user feedback including support for end-users to remove clusters, correct clusters (by adjusting the keywords characterizing a cluster), and correct classifications of individual items within a cluster. In contrast to these interactive approaches, we do not consider systems that automatically cluster documents without user feedback as instances of interactive machine learning (e.g., Zeng *et al*'s system (2004) automatically clusters search results without any user intervention).

Another active area of research in information retrieval involves document annotation (i.e., attaching metadata on documents such as text, images, and webpages). Research in this area has increased recently due to the prevalence and utility of tagging on the Web, which facilitates search and browsing. Automatic and interactive document annotation is of particular importance for improving the quality of human annotations and tags and also to increase the coverage of those tags over untagged documents. For example, several researchers have started to explore tag recommendation to help people quickly annotate documents with better quality tags (e.g., Graham and Caverlee, 2008; Chen *et al*, 2008; Byde *et al, 2007*). Outside of interactive tagging, some researchers have explored interactive techniques for reliably annotating large corpora according to various coding schemes (e.g., Donmez *et al*, 2005; Kulesza *et al*, 2010). For example, Kulesza *et al*'s AutoCoder system (2010) interactively learns how to annotate phrases of text based on some initial manual annotations and then uses its learned model to automatically annotate the remaining data. A user can then iteratively inspect the annotations and correct the system.

2.1.2 Recommender Systems

Recommender systems have their roots in information retrieval, but have become a prominent area of research and development in their own right. The general goal of recommender systems is to select an item or items to present to an end-user that has the most utility for them, where utility depends on the application. For example, utility in product recommender systems might be based on a person's likeliness to purchase a consumer item, whereas utility in news or music recommender systems might be based on a person's computed interest. Unlike search in information retrieval, however, utility is typically estimated via a persistent model representing the end-user themselves, known as a profile or *user model*, rather than a model representing their current and transient information need (Adomavicius and Tuzhilin, 2005).

Early recommender systems often generated user models based on static information gathered once and prior to first use of the system (e.g, Rich and Sidner, 1979; Krulwich, 1997; Kobsa *et al*, 2001; Yan and Garcia-Molina, 1999). For example Rich and Sidner's Grundy system (1979) made book recommendations based on initial dialog with an end-user gathering their demographic information and general interests. We do not consider these systems examples of interactive machine learning because there is no loop over the model being created.

Today, user models in recommender systems are more commonly constructed via some form of interactive machine learning, typically via iterative user feedback analogous to relevance feedback techniques in search. As with relevance feedback in search, end-user feedback in recommender systems is typically provided in the form of explicit item ratings or comments or based on implicit and observed user behaviors (Montaner, 2003).

Recommender systems are generally divided into two classes: content-based and collaborative filtering systems (along with hybrid approaches of the two). Content-based filtering is most similar to relevance feedback because item recommendations are based directly on the content of similarly rated items. Therefore, the interactive machine learning loop is characterized by feedback on items, the combination of those items to generate a model of the user's interests, and then the selecting and ranking of new items based on their similarity to the learned model (e.g., Billsus and Pazzani, 1999; Billsus and Pazzani 2000; Zhang *et al,* 2002).

On the other hand, the interactive loop in collaborative filtering systems is more indirect in that a user's ratings on items are combined to find similar users (i.e., users who rate items the same way) rather than similar items. After similar users are found, actual item recommendations are based on some heuristic extrapolation of ratings from those similar users (e.g., Breese *et al*, 1998; Pennock and Horvitz, 1999; Konstan *et al,* 1997; Billsus and Pazzani, 1998). Therefore, the recommendations here are controlled by a combination of an end-user's input and other people's input (hence the term "collaborative").

Recommender systems are now commonplace in many commercial domains, including music (e.g., Pandora.com, Last.fm), movies (e.g., Netflix.com), shopping (e.g., Amazon.com), and others. In fact, most popular commercial websites offering consumer products now provide some form of recommendations based on an interactively trained user model. However, recommender systems are generally used for low risk tasks (Jøsang and Lo Presti, 2004), particularly due to the lack of transparency and understandability of the modeling and recommendation process. This has led to numerous

investigations into the nature of trust in recommender systems and potential methods for achieving it (see Section 2.2 for a further examination of trust in interactive machine learning-based systems).

Another major issue studied in the research literature pertains to the temporal nature of the user model, including how to best keep the user model up to date or what kind of memory the models should have. For example, in some domains a person's interests or needs change frequently whereas in others interests are longer lived. This has led to some systems employing multiple user models to reflect both short and long term user interests (e.g., Billsus and Pazzani, 1999).

For good surveys on recommender systems see Adomavicius and Tuzhilin (2005), Montaner (2003), and Konstan and Riedl (2012).

2.1.3 Context-Aware Computing

Context-aware computing is one of the major themes in the fields of ubiquitous and pervasive computing which focuses on the seamless integration of computing into our everyday lives (made possible by the ever-increasing proliferation of embedded devices in our physical world). Context-aware computing refers to the sensing and reacting of applications to a person's current context. A person's context can include information about themselves and others who may be near them, their current state or activity, and their surroundings (e.g., where they are and when).

Until recently, most systems and research in modeling a person's context have taken a fixed approach (Davies *et al*, 2008), modeling context as a set of hand-constructed if-then rules or training and fixing machine learned models prior to deployment. Examples include location based services such as call forwarding (e.g., Want *et al,* 1992), tour guide information presentation (e.g., Abowd *et al,* 1997), and contextual reminders or messages (e.g., Schilit *et al*, 1994, Munoz *et al*, 2003). However, in order to support individual users in an increasing number of situations (due to the increasing prevalence of mobile devices and cheap sensors), researchers have more recently begun exploring how to involve end-users and machine learning in the process of creating, using, and adapting contextual models for a variety of applications (e.g., Dey *et al*, 2004; Fogarty and Hudson, 2007; Horvitz *et al*, 2004; Mozer, 1988; Verpoorten *et al*, 2007; Hartmann *et al*, 2007).

For example, Dey *et al*'s aCAPpella system (2004) enables people to specify contexts of interest themselves (as well as actions to take within those contexts), by interactively demonstrating examples of those contexts. A person specifies a context by recording sensor information during real instances of that context (e.g., recording sound, video, and other sensor information during a meeting context). aCAPpella learns a model from these examples which it then uses to automatically take specified actions when it

detects those contexts in the future. A person can continue providing new examples in situ or inspect and edit old examples until the system is performing the way they want it to.

Similarly, Fogarty and Hudson's Subtle system (2007) enables end-users to interactively build sensor-based models of context by example (e.g., by implicitly monitoring a person's behaviors in different situations or by explicitly prompting for labels of contexts). Subtle also supports a sliding window over the learning so as to better keep the models up to date and relevant.

Relevant issues for interactive machine learning in context-aware computing systems include understandability and control of complex sensing and reasoning systems (e.g., Lim and Dey, 2009; Barkhuus and Dey 2003; Bellotti and Edwards, 2001), ambiguity resolution (e.g., Dey *et al,* 2002), and appropriately obtaining accurate context labels and information (e.g., interrupting a user to obtain labeled contextual information can lead to frustration and biased information (Baily *et al*, 2001; Horvitz *et al,* 2004)).

2.1.4 Programming by Demonstration

Programming by demonstration refers to the technique of instructing a computer in how to perform tasks by demonstrating examples of those tasks (therefore, programming by demonstration is also sometimes known as "programming by example"). The general goal of programming by demonstration systems is to enable people to essentially program computers without needing to operate in low-level programming languages. Therefore, the main challenge in programming by demonstration systems is effectively generalizing from a user's demonstration of an intended program's behavior so as to enable autonomous or semi-autonomous execution of that program in the future and in possibly variable situations. Although programming by demonstration is sometimes used as an umbrella term for referring to any interaction in which generalization is made from examples (e.g. *Dey et al* described their aCAPpella system (2004) discussed above as a programming by demonstration system), it is most often is characterized by examples involving sequences of steps or traces (Lieberman, 2000).

Common generalization strategies in programming by demonstration systems include using heuristics or rules to extract patterns from examples and then generating programs from those patterns (e.g., Cypher, 1991; Myers and Buxton, 1986; Modugno *et al*, 1997; Lieberman, 1993; Bergman *et al,* 2005; Spaulding *et al,* 2009). Once generated, these programs can often be viewed and manually edited.

However, because rule-based strategies can result in brittle and limited programs, researchers have also explored interactive machine learning-based strategies for generalization (e.g., Lau *et al*, 2000; McDaniel and Myers, 1999; Lau *et al,* 2004; Garland *et al*, 2001; Paynter , 2000; Paynter and Witten, 2004;

Wolfman *et al*, 2001; Chen and Weld, 2008). For example, Lau *et al*'s SMARTedit system (2000) uses interactive machine learning over demonstrations of text edits to learn a program for performing future edits. The learned program can then be executed step by step, allowing for the interleaving of manual actions with program predictions. When manual actions deviate from the learned program's predictions, the new sequence is interpreted as another example which is then fed back into the model for further generalization. The learned program can also be viewed and manually edited. In another example, Chen and Weld's CHINLE system (2008) enables end-users to interactively train a program for automating repetitive tasks in an interface (e.g., customizing application settings). After demonstrations of a task, CHINLE learns a procedure (or a partial procedure) for automating that task in the future. An end-user can view and edit the system's learned procedure which is displayed as a sequence of color coded steps wherein color intensity indicates CHINLE's confidence in that step. This color coding not only explains the system's current understanding, but can also guide the user towards appropriate edits (e.g., steps in which the system has the lowest confidence). In CHINLE, edits can be made to individual procedures as well as to hypotheses used by the system to generalize from examples. In each case, these edits impact future iterations of CHINLE's task learning.

It should be noted that in heuristic and rule-based programming by demonstration systems, there is also often an iterative loop over the generated program. For example, Cypher's Eager system (1991) monitors a user's behaviors but uses fixed heuristics to detect repetition over those behaviors which it then records as a program. A person can then either validate that the current program is correct (thereby instructing Eager can stop evolving it), or can continue interacting as normal to provide Eager with additional behaviors which it will continue to incorporate into the program. Therefore, the end-user's interaction experience with Eager is similar to that of many interactive machine learning based systems (such as Lau *et al*'s SMARTedit). However, the learned program is inherently different which can impact generalization performance and, in turn, affect the amount of effort and iteration required by the end-user.

Because iteration and program demonstration is expensive (generally more so than providing examples over discrete objects), research issues in programming by demonstration systems include enabling robust generalization from very few carefully chosen examples (e.g., Wolfman *et al*, 2001).

2.1.5 Adaptive and Intelligent Systems

Adaptive and Intelligent Systems are also umbrella terms used to refer to any system that adapts to the needs or abilities of individual users (as opposed to fixed, one-size fits all systems). For example, many of the areas already described in this chapter (e.g., recommender systems, document classification, and programming by demonstration) are sometimes referred to as specific cases of adaptive or intelligent

systems (Jameson, 2008; Ross, 2000). Other examples of intelligent systems include systems that adapt to an individual's skill level (e.g., adapting an interface for accessibility purposes (Gajos *et al*, 2008) or automatically tailoring support or difficulty levels in educational systems (Corbett *et al*, 1997)) or goals (e.g., interface features that are automatically reorganized such that the functions that are most likely to be used next are easily accessible (Mitchell and Shneiderman, 1989; Greenberg and Witten, 1985)). Another class of systems that typically falls under this category is known as interface agents. Interface agents act as personal assistants that attempt to provide proactive support on how to use an interface or carry out tasks on behalf of a user or in collaboration with them (e.g., Horvitz et al, 1998; Maes and Kozierok, 1993).

Not all adaptive and intelligent systems have an interactive machine learning component (e.g., Rich *et al*, 2005; Lesh *et al*, 1999; Lieberman and Espinosa, 2007). For example, Lieberman & Espinosa's (2007) Roadie system uses plan recognition and a pre-defined knowledge base to help people use and debug electronic devices. This system employs a fixed model (not a model learned via interactive machine learning) to aid the user in their task.

When intelligent systems (particularly agent-based system) do employ interactive machine learning, they often embed it within a larger and more complex processing and decision making unit (e.g., Glass *et al*, 2008; Horvitz, 1999). For example, many intelligent tutoring systems include both an interactive machine learning loop (to model the current skill level of the user) as well as a knowledge base of the domain which together influence how the system behaves (Nwana, 1990). Similarly, Hovitz's LookOut system (1999) embeds an interactive machine learning loop (modeling whether emails can be calendered or not) within a larger system which combines the output of this learned model in a decision theoretic process which determines the expected utility of taking action in an interface or not. In such systems, it can be difficult to tease out the effects of an end-user's actions during the interactive machine learning process.

This dissertation focuses on the design of systems that have a more well-defined interactive machine learning loop (such as some of the applications discussed above). However, as discussed in Section 2.2, we can apply some of the insights learned from work on general intelligent systems and agents towards understanding how to design more direct end-user interaction with machine learning.

As in some of the areas of research discussed above, the user model typically plays a key role in adaptive and intelligent systems that employ interactive machine learning (Langley, 1999; Jameson, 2008; Ross, 2000). Therefore, interactive machine learning relevant issues in adaptive and intelligent systems involve generating accurate user models based on interactive user feedback and keeping the learned models up to

date and consistent with an individual's possibly changing needs (e.g., Ross, 2000; Billsus and Pazzani, 2000; Webb, 2001; Koren, 2009; Koychev and Schwab, 2000; Stamou and Ntoulas, 2009). Again, feedback to the user model can be explicit (e.g., based on dialog with the user (Horvitz *et al*, 1998)) or implicit (e.g., based on observed behaviors (Webb *et al*, 2001)). Interestingly, there has also been research in this space on detecting and preventing un-authorized feedback to the system (e.g., detecting when a student is gaming an intelligent tutoring system to advance through the program (Baker *et al*, 2006)).

Other issues include understandability and controllability of complex intelligence and managing expectations (Höök, 2000; Shneiderman and Maes, 1997; Horvitz, 1999; Jameson, 2008). For example, because perfect adaptations are difficult if not impossible to achieve consistently, some have advocated a *mixed-initiative interaction* paradigm in which the user and machine explicitly collaborate towards some goal rather than the traditional approach where the system may take action or adapt completely autonomously (Horvitz, 1999). Because these issues are also relevant to designing for end-user interaction with machine learning, we discuss these further in Section 2.2.

2.1.6 New Uses of End-User Interactive Machine Learning

In addition to the use of end-user interactive machine learning in the established fields discussed above, researchers have also started to explore the use of end-user driven machine learning for new and varied purposes. The following examples push the boundaries of what people can do with machine learning and inspired our own ambitions in this space.

Fails and Olsen's Crayons system (2003) supports the interactive training of pixel classifiers for image segmentation. These classifiers can then be used in camera-based applications (e.g., that visually track objects). To guide the system in learning how to segment images, a person paints over an image indicating both positive and negative regions (i.e., pixels that should be detected and ignored). Crayons uses this information to update its model and then displays its current understanding of how to segment using visual highlighting over the image. A user can then iteratively provide additional examples to correct the system by painting over areas in the image that the system is misclassifying.

Ritter and Basu (2009) employed end-user interactive machine learning to help users perform complex selection tasks (e.g., selecting files to delete or copy or selecting emails to move from large lists). Here a user selects a few examples to start. The system then learns from these examples and automatically selects additional items from the set. A person can continue to iteratively select or deselect items (providing additional positive and negative training examples to the system) until the correct set is selected.

Shilman *et al*'s CueTip (2006) employs interactive machine learning in the process of handwriting recognition. CueTip's handwriting recognizer first interprets some handwritten ink and displays its results. The end-user can then begin correcting the system's interpretation via simple gestures over the results (e.g., crossing out incorrect letters or joining words via a stroke between them). As the user is correcting, CueTip feeds these corrections back into the model for re-analysis. That is, corrections are translated into constraints on the data passed through the recognizer, thereby guiding the system's interpretation.

Fiebrink *et al*'s Wekinator (2009) is a general purpose tool supporting machine learning novices in the interactive training of gesture classifiers. These gesture classifiers are intended for use in creative applications such as musical performances or new musical instruments. With The Wekinator, a person can iteratively demonstrate gestures that they wish the system to recognize (along with sounds they wish the gestures to trigger). The Wekinator then trains a model from video and other sensor data collected during the gesture demonstrations, which it can then immediately apply to detect those gestures in the future. A person can iteratively test the system's recognition performance and correct it when necessary by providing additional gestures or editing previous ones.

Finally, Fogarty *et al*'s CueFlik (2009) is a system for enabling end-users to create reusable image classifiers for automatically detecting visual concepts (e.g., scenic, visually busy, or bright and colorful images). A person can train CueFlik by iteratively providing it with positive and negative example images of that concept while examining the system's current understanding. CueFlik inspired much of our research in end-user driven machine learning and became a test-bed for some of our own work discussed in this dissertation (see Chapter 4).

In this section, we reviewed work in established fields that employ end-user interactive machine learning for domain specific purposes as well as new systems that have started to push the boundaries of what end-users can do with machine learning. This review was not intended to be comprehensive. Indeed, end-user interactive machine learning is also used in other domains such as human-robot interaction (e.g., Goodrich and Schultz, 2007; Nicolescu and Mataric, 2001)) and recognition systems (e.g., Daneu and Dorizzi, 1996; Licsar and Sziranyi, 2005)). This review instead serves to illustrate the diversity of interesting uses of end-user interactive machine learning as well as to provide a reference for designing future systems.

2.2 Exploring How to Design End-User Driven Machine Learning

In this section we discuss work related to understanding how to design end-user interaction with machine learning. Much of the work discussed in this section comes from the adaptive and intelligent systems community and, therefore, considers the question of how to design end-user interaction with systems that incorporate any sort of intelligence (i.e., adaptive and intelligent systems which may or may not use some interactive machine learning, as discussed in Section 2.1.5). Some of this work also comes from specific domains (e.g., designing interaction with recommender, context aware computing, and document classification systems). However, many of the issues raised and solutions proposed in these related works apply to the design of end-user interaction with machine learning and we highlight some of these similarities (and differences) throughout this section when necessary.

The human-computer interaction community has decades of experience in designing traditional systems (e.g., direct manipulation interfaces). This has led to established practices and principles for designing end-user interaction with these systems (e.g., Moggridge, 2007; Dix *et al*, 2004; Winograd, 1996; Norman, 1988). For example, research and practice has shown that systems that support understandability (e.g., systems that are predictable or clear about how they work) and control (e.g., systems that make it clear how a person can accomplish their goals and give them the freedom to do so) are generally more usable than systems that do not support these principles.

However, researchers have pointed out that adaptive and intelligent systems inherently violate some of these tried and tested principles, leading to years of debate about when to use intelligence in systems, if ever (e.g., Shneiderman and Maes, 1997; Lanier, 1995; Norman, 1994; Höök, 2000; Horvitz, 1997). For example, the goal of many intelligent systems is to do some of the work on behalf of the user, potentially violating the principle of control (Norman, 1994). Similarly, intelligent systems often use complex processing which can make it difficult for an end-user to predict or understand the outcomes of the system (Shneiderman and Maes, 1997). Many of these issues are also relevant for end-user interaction with machine learning. For example, a machine learning system evolves as it receives additional user feedback which may make its behaviors difficult to predict. Similarly, machine learning algorithms are inherently complex and therefore might be difficult to control.

Accordingly, researchers have endeavored to identify key challenges for designing intelligent systems (e.g., Höök, 2000; Benyon, 1993; Horvitz 1997). For example, Höök (2000) stresses that the community needs new usability principles for designing interaction with intelligence. Others suggest that we need a better understanding of when intelligence is and is not appropriate (Horvitz, 1997).

Some researchers have also started to suggest new principles for designing end-user interaction with intelligent systems (e.g., Norman, 1994; Höök, 2000; Horvitz, 1999; Jameson, 2008). For example, Norman (1994) and Höök (2000) both identified safety and trust as key factors to consider when designing intelligent systems, referring to the assurance against and prevention of unwanted adaptations or actions. Others have stated that intelligent systems should manage expectations so as not to mislead or frustrate the end-user during interaction (e.g., Norman, 1994; Höök, 2000; Jameson, 2008). In Horvitz's formative paper on mixed-initiative interfaces (1999), he proposed several principles for balancing intelligence with traditional direct manipulation constructs. For example, Horvitz emphasized consideration of the timing of intelligent services, scoping of adaptations or favoring direct control under severe uncertainty, and maintaining a working memory of recent interactions.

A large body of work in the intelligent systems community as well as in other fields that use interactive machine learning has gone on to start exploring solutions for achieving some of these proposed principles or for supporting some of the traditional direct manipulation principles. Here we summarize some of the most explored techniques for designing effective end-user interaction with intelligent and interactive machine learning-based systems. In the next chapter, we analyze these techniques further and discuss other ways that researchers have designed end-user interaction with machine learning.

Explanations have received considerable attention as a design solution for supporting transparency and understandability in intelligent systems (e.g., Bunt *et al,* 2004; Glass *et al*, 2008), recommender systems (e.g., Vig *et al*, 2009; Herlocker *et al*, 2000; Tintarev and Masthoff, 2007; McSherry, 2004; Pu and Chen, 2006; Sinha and Swearington, 2002; Waern, 2004), document classification (e.g., Stumpf *et al*, 2007; Kulesza *et al*, 2010; Kulesza *et al*, 2011; Crawford *et al*, 2001) and context aware computing systems (e.g., Lim *et al*, 2009; Lim and Dey, 2009; Lim and Dey, 2010; Bellotti and Edwards, 2001; Tullio *et al,* 2007).

Herlocker *et al* (2000) designed and evaluated 21 different explanation facilities for collaborative filtering based movie recommendations, including histograms of neighbor ratings (i.e., a summary of ratings on a movie from other users who have similar interests in movies), similarity to already labeled movies (i.e., content based), highlighting important movie attributes (e.g., actors or genres the user is interested), and indicators of past system performance on its predictions. From their evaluation, Herlocker *et al* found that explanations helped increase user acceptance of the recommender system (however, the effects of explanations on end-user decision making was inconclusive).

Lim *et al* (2009) designed ten different types of explanations for context-aware computing systems including why (e.g., why did the system behave the way it did?), why not (e.g., why did the system not behave in some other way?), how to (e.g., how can I get the system to do what I want?), and what if explanations (e.g., how would the system behave if X happened?). From their evaluation, Lim and Dey found that some types of explanations performed better than others at improving user understanding of the system (e.g., why explanations improved understanding while why not explanations lowered it). Lim and Dey (2010) went on to develop a toolkit for aiding developers in generating explanations for future context-aware computing systems.

Stumpf *et al* (2007) compared explanations designed for three different types of machine learning based email classifiers (rule, keyword, and similarity based classifiers) and found that some explanations were more understandable than others (e.g., rule-based explanations were the most understandable while similarity based explanations were least understandable). Stumpf *et al* also found that explanation type impacted the types of feedback that people were able to supply to further guide the system.

Other researchers have explored techniques for influencing end-users' mental-models to improve their understanding and control of interactive machine learning-based systems (e.g., Tullio *et* al, 2007; Borgman, 1986; Muramatsu and Pratt, 2001; Kulesza *et al*, 2010; Kulesza *et al*, 2012). For example, Kulesza *et al* (2012) studied the effects of introductory tutorials on people's mental models of a music recommender system. Their experiment found that tutorials designed to induce a deep understanding of a recommender system's reasoning and behavior resulted in sound mental models and that improvement in mental model accuracy resulted in better ability to control the system. Tullio *et al* (2007) investigated the effects of in situ explanations on mental models of a context-aware computing system. They found that over time, people developed a reasonable understanding of how their system worked, but found that deep mental models were difficult to correct even with explanations.

Explanations and mental models are believed to be effective in improving understandability and control of complex, adaptive systems. However, in lieu of understandability and transparency, particularly when systems are believed to be too complex to explain, researchers have investigated other ways of building trust and acceptance of intelligent and interactive machine learning based systems (e.g., Glass *et al*, 2008; Cramer *et al,* 2008; Cramer *et al,* 2009; Sinha and Swearingen, 2002; Parasuraman and Miller, 2004; Jøsang and Lo Presti, 2004; Alpert *et al*, 2003; Lee and See, 2004; Victor et al, In Press; Dzindolet *et al*, 2003; Muir, 1994; Muir and Moray, 1996). For example, Glass *et al* (2008) investigated factors that people believe impact their trust in an intelligent office assistant that autonomously performs calendaring, emailing, address booking tasks. The authors found that along with inadequate explanations and feedback

(e.g., providing little evidence about whether or not the system was listening to the user), other factors that reduced user trust and acceptance included mismatching expectations about what the system should be able to do with the given information and usability issues such as slow update times and lack of appropriate controls (e.g., undo). Even when people did claim to trust the system, they later revealed that they would still disapprove of autonomous actions without a verification step. Other researchers found that active user participation in the training process (e.g., Cramer *et al,* 2009; Alpert *et al*, 2003), system politeness (e.g., Parasuraman and Miller, 2004), and perceived system performance (e.g., Lee and See, 2003; Cramer *et al*, 2008) all impact trust and acceptance of interactive machine learning-based systems.

Despite these advances, our community still lacks established principles and proven techniques for designing intelligent systems. Accordingly, researchers continue to investigate the causes and suggest new solutions for effective designs. For example, Lieberman (2009) argues that the artificial intelligence and human-computer interaction communities need to work together before we can start to understand how to design effective intelligent systems. In another example, Jameson (2009) proposed a new approach for designing interaction with intelligent systems based around identifying potential side effects of intelligence within different applications and mitigating problems when necessary.

Many researchers have also concluded that, as with traditional systems, the success of an adaptive or intelligent system depends on the context of use and whether or not the system was designed appropriately for that context (e.g., Höök, 2000; Horvitz, 1997; Benyon, 1997; Jameson, 2008; Jameson, 2009; Lieberman, 2009). This suggests that the principles and solutions being proposed need to be weighed in light of the underlying system goals as with traditional interaction design. For example, system transparency might impede efficiency during interaction and therefore consideration must be given to the reason why an end-user is interacting with a machine learning system in the first place (e.g., to quickly accomplish some transient task versus to build an accurate model). Similarly, trust is important when the system will take autonomous action on the user's behalf or when the user is unfamiliar with the data (e.g., deciding on new items to purchase). However, trust may be less critical when the system's predictions will be regulated by the end-user or when the system is operating in a low risk environment (e.g., when automatically selecting songs to play).

Therefore, we argue that what is needed to further advance our understanding of how to design end-user interaction with machine learning is a better understanding of potential end-user goals during interaction along with an understanding of other factors and constraints that might impact the design. By identifying and characterizing these factors, we can better weigh alternative design solutions.

To this end, the next chapter presents our design space for end-user interactive machine learning which can provide a foundation and reference for future researchers and developers of interactive machine learning-based systems.

Chapter 3
A Design Space for End-User Interaction with Machine Learning

In traditional interaction design there is an early focus on the end-user and their goals (Moggridge, 2007; Dix *et al*, 2004; Winograd, 1996; Norman, 1988). The first step in this iterative design process is thus to identify these goals and the constraints that will form the context of the end-user's interaction. Only after these design factors are identified should idea generation begin. Idea generation involves formulating potentially many design solutions for achieving identified goals within the constraints embedded in the task. Finally, design solutions are weighed in light of tradeoffs between competing design factors. This process may continue (i.e., identifying design factors, designing potential solutions, and evaluating tradeoffs) until an appropriate and effective solution is obtained. Goals and constraints therefore directly influence the design choices that system developers can and should make.

In this chapter we present a design space for end-user interaction with machine learning from an interaction design perspective. That is, we examine the end-user interactive machine learning loop itself and distill the following design attributes:

- **Design Factors.** We describe the end-user interactive machine learning process in terms of factors that influence how the interaction between the end-user and the machine should be designed. These factors include potential end-user goals in using machine learning and their contexts of doing so. We also describe constraints that naturally arise from these goals and contexts.

- **Design Dimensions.** We identify and describe design dimensions that have been explored to achieve various interaction goals in various contexts. These dimensions are not necessarily orthogonal and some may intersect. But examining them individually can provide a clearer understanding of the various ways interactive machine learning can be designed.

This design space has been informed by the same body of related work discussed in Chapter 2 as well as our own experiences designing end-user interactive machine learning in several domains (see Chapters 4 to 6). Specifically, we developed this design space by identifying design factors and dimensions explored in existing systems and research. We also abstracted away domain specific details as much as possible to focus on the end-user interactive machine learning loop itself. Note, however, that domain specific factors are partially captured by the context of end-user interaction. We also give concrete examples from existing systems throughout this section to better illustrate the various design attributes.

While domain specific factors do influence how existing systems have been and should be designed, there are many benefits to calling out the interactive machine learning process itself and analyzing it independently. As described in Chapter 2, many fields of research are already employing end-user interactive machine learning to solve domain specific problems. However, these fields often refer to the interactive machine learning process in domain specific terms (e.g., relevance feedback, programming by demonstration) or embed it within larger or more complex systems (e.g., adaptive and intelligent interfaces). This can make it difficult to communicate and learn from one another. Moreover, overlooking design solutions used in related fields could potentially lead to duplicate work. Therefore, by isolating the end-user interactive machine learning process, we can attempt to move beyond ad hoc and domain specific solutions. In turn, this can help accelerate our understanding of how end-user interactive machine learning *should* be designed.

The design space therefore serves the following purposes:

- To provide a common language for understanding and discussing how end-user interactive machine learning has been or might be designed.
- To facilitate a systematic approach towards designing systems that might involve an end-user interactive machine learning loop (e.g., by enabling designers and developers to compare competing design solutions for achieving the interaction goals and contexts identified for their systems).
- To expose new challenges and opportunities for improving the end-user interactive machine learning process.
- To provide a foundation and reference for future researchers and developers of interactive machine learning-based systems.

Note that this design space is meant to be descriptive, not prescriptive. This is for several reasons. First, the design factors and dimensions discussed in this chapter span many diverse fields using different evaluation methods and criteria for gauging success. This makes it difficult to determine which techniques might also be successful in different scenarios. Second, a prescriptive analysis demands comparison of different design techniques in controlled scenarios (e.g., with fixed interaction goals and contexts of use). However, many evaluations in the literature, including some of our own work (e.g., our work on CueT discussed in Chapter 6), have opted for feasibility studies comparing a system with and without an interactive machine learning component. While feasibility evaluations are useful for demonstrating the effectiveness of a system for solving a domain specific problem, it also makes it difficult to attribute the success of that system to any particular design technique employed. For these reasons, it is premature to declare some techniques observed in previous systems as being better than others. However, in Chapter 7 we begin to hypothesize about what some design guidelines might be as a result of our own experiences and our investigation into this design space.

3.1 Design Factors

3.1.1 Interaction Goals and Contexts

In this section we describe interaction goals and contexts that we identified as impacting the design of the end-user interactive machine learning process. Interaction goals refer to the underlying reasons why an end-user might be using machine learning and what they aim to accomplish in doing so (e.g., what their expected or intended outcomes might be). Related to interaction goals is the desired context in which the machine learning will be used to achieve those goals.

Intended Product (Reusable versus Disposable Models)

The *Intended Product* design factor refers to the desired outcome of the end-user interactive machine learning process. The desired outcome could either be the machine-learned model itself or the model predictions (i.e., the output over some data set). Typically, when the *Intended Product* of interactive machine learning is the model itself, the model is intended for *reuse* in the future. On the other hand, if the intended product is the model output, the model itself is essentially a *disposable* side effect.

The *Intended Product* can affect how much time and attention a person should or might be willing to spend on improving the system's understanding. For example, in Fails and Olsen's Crayons system (2003), an end-user interactively trains a model intended for embedding in camera-based object tracking applications. Therefore, the end-user is expected to iteratively correct the system's understanding until the model is working adequately for reuse. In contrast, in Ritter and Basu's file selection system (2009), a person interacts with a machine learning system for the purpose of more quickly select items of interest.

In this case, the user is primarily interested in the output of the model being learned (i.e., the predicted selections) rather than improving the model itself. Furthermore, in Ritter and Basu's file selection system, the current model is only used to help accomplish a person's immediate selection task and future tasks will require entirely new models.

This design factor also has implications on whether or not trust in the system is important (see Section 2.2 for more details on the issue of trust in interactive machine learning). For example, a person intending to reuse a model might desire some guarantees about that model's generalization performance. In contrast, trust might be less critical when the model is disposable and the model's predictions will be regulated by the end-user (e.g., an end-user is expected to verify selections by Ritter and Basu's file selection system (2009)). However, the need for trust is also dependent on other design factors that we identify below (e.g., performance requirements, volume of data). For example, trust might only be important for reusable models that will be used in high risk situations (e.g., email classification), and less for reusable models intended for lower risk tasks (e.g., recommender systems). Similarly, trust might be still be important for disposable models operating over large data sets where it may be more difficult or infeasible to verify all of the model's predictions (e.g., with Kulesza *et al*'s AutoCoder system that helps people annotate a large corpus of text (2010)).

Interaction Focus (Interaction with Machine Learning as a Primary versus Secondary Task)

Interaction Focus refers to whether a person's attention during interaction will be on the machine learning process or on some external or domain specific goal (and the interaction with the machine learning is simply a means for accomplishing that goal).

When an end-user's *Interaction Focus* is on the machine learning, they will typically dedicate some time and attention to working with the system. For example, a person's focus during interaction with Kulesza *et al*'s AutoCoder system (2010) is to guide the machine learning to annotate a large corpus of text. Therefore, the end-user expects to continue collaborating with the system until the text is adequately annotated. In contrast, with Ritter and Basu's file selection system (2009), a person's focus is on selecting items. In this case, the machine learning can work in the background to help but can also be ignored.

The *Interaction Focus* design factor can also impact how rich the interface to the interactive machine learning process can be as well as how tolerant a person might be to more aggressive guidance. Again, because a person's focus with Kulesza *et al*'s AutoCoder system is on collaborating with the machine learning, the system could display a considerable amount of information to help the user more effectively interact with the machine learning (e.g., explanations, confidence metrics, impact counts, a history of

prediction changes). In contrast, the interface to Ritter and Bau's file selection system is appropriately much less complex. Furthermore, in other cases in which a person's focus is on something other than the machine learning process, the person may not even be aware that they are interacting with a machine learning system at all (e.g., recommender and relevance feedback systems that use implicit observations of user behavior to improve the model).

Note that *Interaction Focus* is related to but different from the *Intended Product* design factor. For example, in Kulesza *et al*'s AutoCoder system, interaction with the machine learning is a person's primary task but the model is disposable. Whereas with Fail's and Olsen's Crayons (2003), interactive machine learning is again a person's primary focus, but in this case they are working towards creating a reusable model. Correspondingly, interaction with the machine learning is secondary in Ritter and Basu's file selector (2009) and the model is disposable. However, with recommender systems, interaction is again secondary (because a person is focused on evaluating recommended items) but the underlying model is intended for prolonged use.

Evolutionary Needs (Ranging from Fixed to Evolving Models)

The *Evolutionary Needs* design factor refers to whether or not the model, once in a functional state, is intended to be fixed or needs to adapt to potentially changing end-user needs or situations of use. This design factor is a spectrum ranging from fixed models with no evolutionary needs to models that need to frequently update. Typically, models that do not need to evolve are capturing concepts that are somewhat objective and stable. In contrast, models that should evolve over time typically capture subjective interests or needs that might change as a result of new data (a phenomenon known as concept drift).

A model's *Evolutionary Needs* affects how frequently a system needs to update the model as well as when and how often a person might be required to provide additional guidance. Fiebrink *et al*'s Wekinator (2009) is an example of a system in which a model is intended to be fixed once created because gestures are relatively stable concepts (e.g., a gesture is either a hand wave or not). At the other end of this spectrum, Billsus and Pazzani's news recommender (2000) includes both long and short term user models. The long term model evolves slowly to capture a person's general interests. However, the short term model continually evolves to keep up to date with a person's rapidly changing interests (which often depends on new data that may be arriving from sources such as breaking news stories).

Evolutionary Needs also impacts the new user problem in some interactive machine learning systems (e.g., recommender systems, intelligent interfaces). The new user problem occurs when the system does not have enough information about an end-user to make reasonable adaptations or predictions for them.

To mitigate this potentially frustrating experience, some systems initially employ a canonical user model and then slowly adapt that model based on interaction with new users (Langley, 1999; Ross, 2000).

Note that even concepts that are thought to be relatively stable might benefit from a model capable of evolving. Consider Dey *et al*'s aCAPpella system (2004) which models end-user defined contexts such as meetings. Meetings are relatively objective concepts. However, because machine learning systems are designed to generalize from a limited set of examples, it is likely that not every scenario has been accounted for. In this case, the model might benefit from being able to incorporate data from new situations so as to improve over time.

Concept Flexibility (Well-Defined verses Open-Ended Concepts)

Concept Flexibility refers to whether a person will have a *well-defined* concept in mind prior to interacting with the machine learning or whether that concept might be flexible and more *open-ended*.

The *Concept Flexibility* design factor can affect the type of feedback that should be displayed by the interactive machine learning system such as whether or not to organize system predictions to display diversity in results (see Section 3.2 below). This design factor also affects whether or not system predictability is important during the interaction. For example, in recognition systems like Fiebrink *et al*'s Wekinator (2009), a person typically has a clear concept in mind that they are attempting to convey or teach the system (i.e., a particular gesture). In this case, predictability is important and the system feedback should focus on presenting matching gestures. In contrast, in search and browsing systems, a person's conceptual target might be less clear or more exploratory in nature. Here, because a person's intended concept might evolve as a result of interacting with the system, the system might display a variety of results that a person may have not even anticipated but still might be of interest.

Performance Requirements (e.g., Accuracy and Efficiency)

This design factor pertains to any model or system requirements resulting from the context of use. For example, model *accuracy* might be important in high-risk scenarios where errors are costly. In contrast, *efficiency* might be desired over accuracy in low-risk situations. In other scenarios, *precision* in model predictions might be preferred over *recall*.

Performance Requirements can affect the design of interactive machine learning in many ways. A person might be more willing to spend time and effort training or debugging a system in high-right scenarios where accuracy is critical. For example, it is generally considered critical for email classifiers to be accurate because missing important emails (e.g., about upcoming work meetings) can result in high costs to the end-user. In this situation, a person might benefit from feedback about a system's estimated

accuracy or confidence in its predictions. Alternatively, a person might prefer verifying a system's output in high risk situations rather than allowing the system to take autonomous actions on their behalf (as in Glass *et al*'s 2008 study of trust in an intelligent office assistant).

Model Ownership (Personal or Shared Use)

When creating reusable models, *Model Ownership* refers to whether those models are intended for personal use or might be shared with others. Most previous systems and research have focused on models intended for personal use. However, we expect shared models to become more commonplace in the future. For example, as interactive machine learning becomes a reality in our everyday applications, people will likely want to make use of already created models rather than starting from scratch. People might also want to share their classifiers socially (e.g., a person might want to share their musical interests with their friends via a trained music classifier).

Designing interaction with models intended to be shared might affect how much attention a person should pay in assessing a model's generalization performance on new data sets. *Model Ownership* could also impact how models might be described by their creators or searched for and adapted by their future users for new situations and purposes.

3.1.2 Constraints

Constraints refer to contextual factors that are less flexible or inherent to the task and identified interaction goals. Here we discuss three general categories of constraints that we identified as impacting how interaction can be designed: the data, the target end-users themselves, and the interaction environment.

Data Constraints

Nature of the Data

This constraint refers to characteristics of the type of data that the end-user and the machine learning will need to operate over during the course of the interaction (e.g., text, images, multimedia, sound, sensors).

The *Nature of the Data* shapes many aspects of the interactive machine learning process. For example, the data type dictates the features that can be used to represent items in the machine learning algorithm and whether or not any features might have missing values. The data type also affects how the examples or individual features might be displayed to the end-user. For example, text and images might be straightforward to present, whereas sensor data might require custom transformations before they can be presented to an end-user.

The diversity of features might also affect how straightforward examples or features might be to display. For example, data types comprised of heterogeneous features might require multiple custom transformations (e.g., emails might be represented by text, time of day, and sender features, all of which might occupy different areas on screen and might require different display techniques). The data type might in turn enable or prohibit interaction over the different elements displayed (see Section 3.2 for more details on possible feedback and control types).

Data Source

The *Data Source* refers to from where the data is originating. For example, the data source may be static (i.e., pool-based) or streaming (i.e., where new data is constantly arising). The source could also be intermittent in that new data is arriving but more sporadically. Data could also be originating from multiple sources (e.g., multiple sensors collecting data in a context-aware computing system).

This constraint effects when and how often a person can or should interact with a machine learning system. The data source also impacts whether a model will be operating on the same or similar data from which it was trained (e.g., with the same distribution) or will need to operate on dissimilar data. The *Data Source* may therefore necessitate a design that supports verification or evaluation of generalization performance during the training process.

Volume of Data

This constraint refers to the amount of data over which the end-user and machine learning system is expected to operate.

The *Volume of Data* can impact how much data a person might be required to process or view at any given time. This, however, can also be affected by a person's *Interaction Focus* or *Performance Requirements* of the system. For example, sampling could be used over large data sets if a person's focus is limited or the system is operating in a low risk environment.

End-User Constraints

Machine Learning Experience

This constraint refers to how much experience an end-user has using interactive machine learning systems. Experience also relates to an end-user's mental model of how an interactive machine learning system works (see Section 2.2 for a more detailed discussion on mental models of interactive machine learning).

Machine Learning Experience can affect a person's ability to effectively train a machine learning system. For example, a person with more experience might require less guidance in steering a system than a person unfamiliar with the interactive machine learning process.

Domain Expertise

Domain Expertise refers to how familiar an end-user is with the domain or data over which the system will operate. A person might also be considered a domain expert when the data involved is easy to inspect (e.g., some types of images).

This constraint affects how easy it is for a person to make decisions on items for feedback to the machine learning or for some external goal (e.g., purchasing items). For example, a person training a personal email classifier will be operating over known data or data that is easy to inspect. In contrast, during search and in some recommender systems a person might be being shown items that are unfamiliar to them. Therefore, this constraint affects how data should be displayed or what information should be presented to the end-user for effective understanding and use of the system.

Motivation

Motivation refers to incentives that people might have in interacting with a machine learning system. For example, a person might be interacting with a system for their own personal gain (e.g., to create a model for personal use). Alternatively, a person might be interacting with a machine learning system as part of their job (e.g., an operator employed to train a system).

This constraint can impact how engaging the interface or interaction needs to be. For example, a person not intrinsically motivated to supply accurate information to the machine learning might require additional incentives to keep them interested or alert.

Individual Differences

This constraint refers to other characteristics that an individual might possess and that may impact the design of a system (e.g., gender, personality traits).

This constraint can affect how aggressive or confident a person might be in steering the system. For example, Kulesza *et al* (2011) found that people with higher self-efficacy were more confident in their feedback decisions and were more willing to experiment during interaction with their AutoCoder system.

Collaboration

This constraint refers to whether a single person will be interacting with the machine learning system at any given time or whether a group or multiple people are somehow contributing to the learning.

Collaboration has received little attention in previous work, but might become more relevant in the future (e.g., as advances are made in the areas of human computation and crowdsourcing).

Collaboration can impact how interaction must be coordinated between multiple people to achieve some common goal. For example, Lau *et al*'s Sheepdog programming by demonstration system (2004) enables multiple domain experts to create technical support procedures via a pipeline workflow (i.e., in which successive experts can expand a procedure created by preceding users). Similarly, Hoffman *et al'*s Kylin information extraction system (2009) operates on Wikipedia articles and receives in situ feedback about its text extractions from multiple users. This feedback is then used to improve future extractions. Other strategies for coordinating multiple users might involve a divide and conquer approach in which portions of data are distributed amongst multiple end-users and then recombined for training.

Environmental Constraints

Display Size

Display Size refers to how much screen real estate might be available for the end-user and machine to communicate over during the interactive machine learning process, if any.

The *Display Size* can impact the amount of feedback the system can provide the end-user during interaction. This constraint can also impact the type of end-user control that can be achieved. For example, Billsus and Pazzani's news recommender (2000) came in two varieties, a desktop and mobile version. The desktop version could display the entire ranked list of news recommendations to the end-user, whereas the mobile version only displayed the top few recommendations. The desktop version also supported explicit feedback through interface controls (e.g., 'interesting', 'not interesting', 'known' buttons), whereas their mobile version learned exclusively from implicit behaviors.

Surroundings

This constraint refers to any characteristics of an end-user's expected physical location during interaction with a machine learning system.

An end-user's expected *Surroundings* can impact their attention to training as well as where and when feedback or guidance is required. For example, some context-aware computing systems can only receive in situ guidance from end-users. In such cases, care must be taken to display feedback or request guidance appropriately (e.g., so as to avoid untimely interruptions).

3.2 Design Dimensions

In this section, we present several dimensions outlining how previous systems and research have designed end-user interaction with machine learning (for various interaction goals and contexts of use). These design dimensions can help us systematically navigate the space of interaction techniques that have been previously explored. In addition, these dimensions can potentially expose open areas of research and new opportunities for design.

We loosely group these dimensions into three high-level categories representing the fundamental characteristics of the iterative cycle between an end-user and a machine learning system: system feedback, end-user control, and temporal. Neither the categories nor individual dimensions here are necessarily independent. For example, the feedback technique supported can influence the control techniques that are achievable. Similarly for individual dimensions within a category. However, loosely categorizing our dimensions in this way facilitates understanding and discussion of this space.

3.2.1 System Feedback

System feedback refers to how the system illustrates its current understanding of the concept being trained. This category generally corresponds to how the machine learning can communicate to the end-user.

Feedback Type

Feedback Type refers to which elements of the interactive machine learning process the system is communicating to the end-user about. These elements can include the following:

- ***Output:*** refers to feedback about the learned model's predictions on the data (e.g., recommender and relevance feedback systems present items the system predicts might be of interest to the user). *Output* feedback can also include information about the system's confidence in its predictions or understanding (e.g., Herlocker *et al,* 2000; Kulesza *et al*, 2011; McNee *et al*, 2003). *Output* feedback is the most common form of feedback in existing interactive machine learning systems because it hides much of the complexity of the underlying algorithm.

  The organization of *Output* feedback has also been well-studied in the literature. For example, previous work has compared organization techniques for displaying recommendations and relevance feedback lists including ranked lists, n-best lists, and clusters to show diversity of feedback (e.g., Pu and Chen, 2006; McGinty and Smyth, 2003; Gass *et al*, 2009; Harrison and Klein, 2007; McSherry,

2004). Other organization techniques include in situ predictions (e.g., Fails and Olsen's Crayons (2003) shows pixel level predictions via highlighting over an image).

- ***Input:*** refers to feedback about items or information that has been provided to the system and is impacting its learning. For example, some systems can present an end-user with a history of observations that are affecting its model of the user. *Input* feedback also includes provenance information about what input data has affected the current prediction (e.g., Billsus *et al*'s FXPALBar (2005) highlights words on a page being viewed that are triggering the current recommendation).

- ***Features:*** refers to feedback about the features being used in the machine learning. This includes what features the system might know about or what features it might be computing as important (e.g., by displaying weights on features as in Kulesza *et al* (2011)).

- ***Logic:*** refers to feedback about the machine learning's internal processing. For example, Lim and Dey's system (2010) generated explanations by displaying paths through a decision tree taken to come to the current predictions. In another example, Herlocker *et al* (2000) examined feedback revealing the nearest neighbors identified and used to infer recommendations in their collaborative filtering system.

- ***Processing State:*** refers to information about the state of the machine learning's processing, independent of the inner workings of the algorithm. For example, Maes and Kozierok's intelligent personal assistant (1993) uses facial expressions on an animated agent to convey different processing states of the system (e.g., an "alert" expression conveys that the system is observing the users actions and a "thinking" expression conveys that the system is in the process of computing a prediction).

Feedback Type is a recognized and relatively well studied dimension in the end-user interactive machine learning literature, particularly in regards to designing for transparency and understandability (see Section 2.2). Often *Feedback Types* are discussed in terms of whether the feedback is about the inner workings of the machine or is independent of the system's underlying reasoning (i.e., white or black box feedback, respectively).

Feedback Granularity

Feedback Granularity refers to the level of detail over a *Feedback Type* presented by the system to illustrate its understanding. *Granularity* can range from coarse- to fine-grained feedback. Examples of coarse-grained feedback include aggregations, summative visualizations, and abstractions (e.g., summaries of model predictions or quality as in our CueT system discussed in Chapter 6). Examples of

finer-grained feedback include presenting some or all of the actual output data (e.g., ranked lists of items in relevance feedback systems) or features (e.g., Kulesza *et al,* 2011).

Feedback Variety

Feedback *Variety* simply refers to the range of feedback types presented in an interface. This dimension ranges from a single feedback mechanism (e.g., a ranked list in relevance feedback systems) to more complex systems presenting multiple views over the data (e.g., our work on CueFlik discussed in Chapter 4 presents multiple views and histories over the interactive machine learning).

Update Visibility

This dimension refers to how apparent the effects of a person's actions are during the interaction process or how obvious it is that there has been some kind of update to the system. For example, in the CueFlik system (2008), every new example image provided by an end-user results in a model update and this update is made explicit via a reshuffling of the data. On the other hand, the effects of new ratings on a user model are typically not immediately observable in most recommender systems. However, it is often implied that these systems have incorporated the new user input in some way (e.g., some systems display a polite "thank you" to acknowledge that the input has been received). In other cases, feedback maybe entirely hidden from an end-user, particularly when system designers do not intend or want the user to be aware that they are being observed. For examples, some intelligent tutoring systems might want to hide feedback about the learning process in order to prevent users from gaming the system (Baker *et al*, 2006).

Push versus Pull Feedback

This dimension refers to whether feedback is offered or presented automatically by the system or whether it must be requested by the user. In some cases, a variety of feedback mechanisms are present but at different levels of availability. For example, Lim and Dey's context aware computing system (2009) will predict and present detected activities but a person must request additional details (e.g., why explanations of system behavior).

3.2.2 End-User Control

End-user control refers to the means by which an end-user can guide or influence the machine learning's behavior and understanding. Therefore, this category corresponds to directed communication from the end-user to the machine.

Note that some of the control dimensions identified here directly correspond to system feedback dimensions (e.g., granularity, variety). However, system feedback techniques do not always dictate what end-user controls are available. In fact, much of the research on system feedback has focused on one way

communication from the machine to the end-user (e.g., classifier explanations that do not allow manipulation or editing as in Herlocker *et al*, 2000, Lim *et al,* 2009 and Tullio *et al*, 2007). In these cases, separate control mechanisms are provided for guidance to the learning. We therefore consider system feedback and end-user control as separate categories.

Control Type

Control Type refers to what elements of the machine learning the end-user can manipulate to guide the system:

- ***Input:*** includes selecting and labeling the examples from which the system will learn. This is the most common form of control and is the technique used by most real-world interactive machine learning systems (e.g., ratings in recommender systems). This dimension also includes the creation or demonstration of new examples (along with their labels) such as in programming by demonstration and context-aware computing systems.

- ***Output*****:** refers to control over model output or predictions that are not fed back into the learning process. Examples include manual editing of a learned program in a programming by demonstration system before it is used for automation and approving an intelligent system's classifications before automated actions are taken (e.g., approving of email predictions before filing).

- ***Features****:* refers to control over the features the system might use to represent the data. This includes control over feature generation, selection, and weighting (e.g., Druck *et al*, 2009; Raghavan and Allan, 2007; Wong *et al*, 2011). For example, Wong *et al* (2011) allowed end-users to explicitly define relevant features for a document classifier by highlighting words in a page. This also includes interaction with features for the purpose filtering the input or output of the model data (e.g., Secord *et al*, 2010, our ReGroup system discussed in Chapter 5).

- ***Classes:*** similar to control over features, this refers to the creation, destruction, or manipulation of classes used by the system. For example, in Hearst *et al*s Scatter/Gather system (1995), a person can select clusters of interests and remove irrelevant ones. Similarly, with our CueT system (see Chapter 6), a person can create new classes for the system to use in making future recommendations. This dimension also includes control over class boundaries (e.g., Ware *et al*, 2001, Kapoor *et al,* 2010).

- ***Parameters:*** refers to control over any other parameters of a machine learning algorithm. Few end-user systems have provided controls over algorithm parameters because of the recognized complexities of such algorithms, particularly for people with little to no machine learning experience

(Norman, 1994). One example is Ware *et al*'s (2001) system which provides control over the parameters of a decision tree learner (e.g., by interactively specifying when and how to split a node) and was targeted at novice users with little machine learning expertise. Similarly, Fiebrienk *et al's* Wekinator (2009) enables users to inspect and configure different algorithm parameters, but again is targeted to machine learning novices.

Control Granularity

Control Granularity refers to the level of control over instances of a type. For example, input labels can be coarse or fine grained (e.g., likes or dislikes versus continuous rating scales in recommender systems (Amoo and Friedman, 2001; Swearington and Sinha, 2002; Cosley *et al*, 2003)). Labels can also be directional (e.g., dynamic critiquing in recommender systems as in McCarthy *et al*, 2005). Input examples themselves can also be provided to the system in varying quantities. For example, in most recommender systems items are labeled one at a time. In contrast, Fails and Olsen's Crayons (2003) allows end-users to quickly label multiple pixels in as single broad brush stroke over an image.

Explicit versus Implicit Control

This dimension refers to whether a person is explicitly providing guidance to the system or whether it is happening passively and potentially without user awareness. For example, object labels and ratings are explicit forms of control, whereas relevance feedback based on behavioral observations are implicit. This is a well-studied dimension in interactive machine learning research (e.g., in research on relevance feedback, recommender systems, and adaptive systems).

Mandatory versus Elective Control

This dimension refers to whether or not control is required from the user at any point during the interaction process. For example, in traditional active learning, the system may not continue operating or evolving without explicit user input. In contrast, user feedback is typically optional in commercial recommender systems (although often encouraged and incentivised), so the user can choose if and when they want to provide it.

Guidance

When controls are provided, this dimension refers to the amount of direction the system provides the end-user in steering the system. *Guidance* can come in many forms, including system suggestions of what to do next or interface scaffolding that restricts or influences what a user can or cannot do at any given step.

Guidance is a well-studied dimension in the interactive machine learning literature. For example, active learning is a form of guided control in that the system solicits specific information from the end-user (e.g.,

labels). This information is then used to update the underlying model. A more flexible example is that of Wolfman *et al*'s SMARTEDIT system (2001), which suggests or requests certain types of examples which the user can provide or ignore in a mixed-initiative fashion. Another example of this mixed initiative approach to guidance is Kulesza *et al*'s What You See is What You Test (WYSIWYT) method of prioritizing and highlighting system predictions that are likely to be wrong in their AutoCoder text annotation system (2011). Here, the user is directed to certain examples which they can either fix or ignore and find their own examples to correct. The AutoCoder system also takes an additional step of propagating user labels (corrections) to nearby examples, thereby also taking part in controlling the machine learning.

In some cases, guidance or restrictions are unintended but result as an artifact of system feedback. For example, recommender systems that present n-best lists restrict the examples a person can provide to further guide the system. As a result of this, some researchers have explored alternative ways of presenting examples that provide more diversity in the recommendations presented and therefore provide more options for control (e.g., McGinty and Smyth, 2003; Pu and Chen, 2006). Other examples of potentially unintended guidance are systems that probe the user for feedback with questions that might influence their response (e.g., questions like "is this relevant to you" and "do you like this" might elicit different responses). In fact, Cosley *et al* (2003) showed that recommendations themselves (and the explanations used to promote them) can influence a person's responses. Therefore, it might be argued that few systems provide complete freedom in end-user control. However, some come close. For example, relevance feedback systems that provide access to all of the data might be considered unguided, but because these systems most often order the data based on similarity or confidence, they still influence the control provided.

Control Variety

As with *Feedback Variety*, this dimension simply refers to the variety of end-user controls presented in the interface. For example, some recommender systems might provide both explicit and implicit controls over the learning or multiple forms of explicit control (e.g., ratings and comments on items).

Direct versus Indirect Control

This dimension refers to the level at which the system adheres or appears to adhere to a user's feedback. This dimension is typically dependent on the choice of machine learning algorithm used and the complexity of the system in which the interactive machine learning is embedded. For example, in Stumpf *et al*'s (2008) email classifier, end-users were allowed to adjust weights on individual features used in the learning algorithm. However, they became frustrated when these seemingly simple adjustments did not

have the desired effect on the system's behavior (which was due to the fact that system updates were based on a combination of the user provided weight and the system's internal weighting). To avoid such frustration, some researchers have designed techniques to more directly incorporate user feedback in interactive machine learning systems (e.g., Kulesza *et al*, 2011; Kapoor *et al*, 2010). For example, Kapoor *et al*'s ManiMatrix (2010) attempts to directly adhere to end-user control adjustments (which can be "locked" by the user to emphasize their expectation that the system should obey their feedback) and notifies the user when their adjustments cannot be achieved due to complex dependencies in the algorithm.

3.2.3 Temporal

The temporal category is concerned with dimensions relating to time, such as when interactions might occur between an end-user and a machine learning system and how often.

Note that the temporal dimensions discussed in this category are sometimes dictated by the context of system use. For example, an inherently sporadic *Data Source* might dictate when system feedback can be made available (see *Timing of System Feedback* below) or when end-user control is obtainable (see *Timing of End-User Control* below). In these cases, temporal dimensions might be considered more like constraints that must be met (i.e., design factors) rather than flexible design decisions. However, in many cases observed in the literature (as will be discussed below), temporal aspects have been manipulated by system designers and therefore we present them here as design dimensions.

Timing of End-User Control

This dimension refers to when and how often end-user control is provided. For example, some adaptive and recommender systems require user feedback up front before they can make any initial predictions. These systems also typically accept ongoing user feedback.

This dimension is often dependent on control *Guidance*. For example, sometimes a system might demand feedback in certain situations or at specific intervals (e.g., Tullio *et al*, 2007; Horvitz *et al*, 2004). In other cases, the user determines the timing of control (e.g., elective and in situ feedback in recommender and context-aware computing systems).

Timing of System Updates

This dimension refers to when and how often the system updates its model. The *Timing of System Updates* could be driven by the end-user or the system itself. For example, an end-user could explicitly tell the system to update (e.g., via an 'Update' button). Alternatively, a system might only update at scheduled intervals or after a specific number of examples have been provided (Ghorab *et al*, 2011).

System Memory

System Memory refers to how long end-user feedback affects the behavior of the system. For example, many user model-based interactive machine learning systems include a policy for forgetting user feedback or a sliding window in order to better model potentially changing user interests or needs (e.g., Crawford *et al*, 2001; Fogarty and Hudson, 2007; Montaner *et al,* 2003).

Control Direction

This dimension refers to the direction of end-user control supported in the interactive machine learning cycle. End-user control in the forward direction includes enabling end-users to supply additional item labels. Backward control typically refers to whether or not revision or reversal of actions is supported.

Most interactive machine learning systems primarily support forward interaction. Our work on supporting model exploration in CueFlik is one example where multiple control mechanisms are provided for both forward and backward control (see Chapter 4 for more details).

Feedback Direction

Analogous to control direction, this dimension refers to whether system feedback is reflective or predictive (i.e., displaying what did happen versus what will happen or both).

Most interactive machine learning systems are reflective, displaying the current prediction which is typically the result of incorporating the most recent end-user interaction into the model being learned. Other mechanisms for reflective feedback include showing progress (e.g., Kulesza *et al,* 2011), showing change in model predictions (e.g., Stumpf *et al,* 2008), and showing histories of update stages (e.g., our work on model exploration in CueFlik discussed in Chapter 4).

Less common is feedback in the forward direction which might show a preview of how the model or system will change as a result of a potential user action. Some examples of forward feedback include explanations supporting "what if" type questions which display how the system will behave as a result of alternative input as well as "how can I" questions which explain how an end-user might achieve a desired outcome (e.g., Lim and Dey, 2009).

One reason predictive feedback is rare is because of the computational expense of computing alternative outcomes, particularly over large datasets. However, previews could help end-users provide more effective control (e.g., by helping people select appropriate actions for achieving desired outcomes (Kulesza *et al,* 2011)). Therefore, more research in reducing computational costs might be valuable, such as previewing approximations of what might happen or preloading expected alternatives.

Human Learning

Human Learning refers to an end-user's potentially evolving mental model of how an interactive machine learning system works. As discussed in Section 2.2, an end-user's mental model can be influenced by the design of a system (e.g., by tutorials (Kulesza *et al,* 2012) or system feedback (Tullio *et al*, 2007; Borgman, 1986; Muramatsu and Pratt, 2001; Kulesza *et al*, 2010; Kulesza *et al*, 2012)). Therefore, care must be taken on this dimension so as to better manage end-user expectations and enable effective guidance during the interactive machine learning process.

3.3 Discussion and Limitations

This chapter presented a novel design space for end-user interaction with machine learning. In particular, we identified and discussed factors affecting the end-user interactive machine learning loop and outlined dimensions that have been explored in the design of existing systems and research.

It should be noted that not all of our identified design factors must be considered when designing a new end-user interactive machine learning system. For example, the *Evolutionary Needs* design factor often only applies after model deployment (e.g., after a reusable model is developed and embedded in an application). Therefore, this design factor would not typically affect systems involving disposable models or models intended to be fixed once created (see our work with CueFlik in Chapter 4 for an example). Instead, this design space is intended to give researchers and developers insight into what factors might contribute to the success or failure of a new end-user interactive machine learning system and to assist and inspire them during design by outlining dimensions considered in previous systems.

We also do not assign an importance value on any design factor presented in this chapter as this is generally dependent on domain specific factors affecting the context of use of the end-user interactive machine learning. However, we do encourage assessment of the impact of relevant design factors as well as the prioritization of these in order to help evaluate tradeoffs on potential design solutions. For example, a prioritization of reusability (i.e., the *Intended Product* of interaction) and model accuracy (i.e., *Performance Requirements* of the system) over the desired *Interaction Focus* might lead to a design that favors accurate assessment via detailed *Feedback Granularity* and control *Guidance* over a less intrusive or simplistic interaction.

This design space was informed by many related works involving end-user interactive machine learning (as discussed in Chapter 2) as well as our own experiences designing end-user interactive machine learning systems (discussed in Chapters 4 to 6). However, as discussed in Chapter 2 on Related Work, many other fields have used or are starting to use some form of end-user interactive machine learning in

their systems and research (e.g., human-robot interaction, natural language processing, data mining). Therefore, we do not claim this design space to be exhaustive. On the contrary, examining other systems and research might reveal additional factors and dimensions that should be considered when designing end-user interactive machine learning systems.

In addition, this design space was not developed via a formal analysis (e.g., we did not take a grounded theory approach). However, a formal examination might improve upon the design space as presented. For example, a systematic review of all related end-user interactive machine learning-based systems by multiple independent coders might reveal additional design factors or might eliminate, combine, and verify factors and dimensions presented here. We therefore advocate such an analysis as future work (see Chapter 7).

Finally, as mentioned in the beginning of this chapter, this design space is intended to be descriptive, not prescriptive. In Chapter 7, we begin to hypothesize about how to design end-user interactive machine learning with respect to various design factors presented here and suggest opportunities for future research.

Chapter 4
Web Image Classification with CueFlik

The ubiquity of image capture devices along with the desire and ease of information sharing has led to an explosion of images available on the Web. However, searching and filtering through these vast image stores remains difficult because image search is still largely based on keywords, despite the fact that images typically lack the metadata to support keyword based sorting and searching. For example, Web image search is typically based on keywords taken from Web pages, photo sharing sites rely upon tagging, and personal photo software typically uses only automatically captured metadata (e.g., time-stamps and GPS). Such information is insufficient for visually characterizing images (visually different images may have the same keywords, and visually similar images may have very different keywords). Consider a person who issues the query "beach" at the photo sharing site Flickr, hoping to find *scenic* pictures of natural beaches taken from above during the day. Such a query is well beyond the capabilities of widely-deployed systems, so the person will be left to manually sift through many irrelevant images (images of sunsets seen from the beach, people partying in beach houses, and hotel-lined resort beaches).

To aid end-users in image search, experts and developers have manually created several image classifiers for use as automatic image filters in search engines. For example, many popular search engines such as Google.com and Bing.com provide preset filters for visual concepts such as photographs, clipart and faces. However, there is currently no *scenic* concept available for automatic image filtering in major search engines. Moreover, it is impossible for developers to anticipate all of the visual concepts people may ever be interested in searching for.

CueFlik is an end-user interactive machine learning application that was developed to enable end-users to create reusable image classifiers themselves (Fogarty *et al*, 2008). These classifiers can then aid end-users during Web image search by re-ranking keyword-based search results according to the visual characteristics learned by the classifier. For example, after issuing a keyword query for "beach" in

CueFlik, a person can provide a few examples of the desired *scenic* images and a few examples of non-scenic images in order to create a *scenic* classifier. Once trained, the classifier can then be applied to sort *scenic* images to the top. Furthermore, this *scenic* classifier can then be applied not only to "beach", but also to other search queries such as "mountain", "water", or "sky". Whenever applied, the *scenic* classifier helps to quickly find images with the same visual qualities as the examples used to train it.

CueFlik became our first test bed for exploring new ways of designing the end-user interactive machine learning process. In this chapter, we first describe the state of the art and related work in image search and classification. Next, we summarize fundamental aspects of the CueFlik system itself. The details of the original CueFlik system can be found in Fogarty *et al*, 2008. We then discuss the design factors affecting end-user interaction with CueFlik with respect to our design space presented in Chapter 3. These design factors influenced our explorations in designing end-user interaction with CueFlik. These explorations targeted three important aspects of the interactive machine learning process: (1) illustrating the system's current understanding of a learned concept (Amershi *et al*, 2009), (2) guiding end-users to select effective training examples (Amershi *et al*, 2009), and (3) enabling end-user exploration of multiple potential models (Amershi *et al*, 2010). We conclude with a summary of our work on designing end-user interaction with CueFlik.

4.1 State of the Art in End-User Driven Image Search and Classification

The current state of the art in image search on the Web mostly involves a combination of keyword search and filtering based on predefined concepts such as photos, faces or clipart. Many popular search engines have also recently started to provide a feature for finding similar images (e.g., the "Show similar images" option in Bing.com and "Similar" option in Google.com). This feature enables a person to pivot on a particular image to display additional images that are most similar to the pivot image (i.e., the nearest neighbors of that image). In contrast, CueFlik enables an end-user to train a reusable model for recognizing a class or type of image.

More similar to CueFlik are technologies such as Apple's iPhoto and Facebook's tag suggestions feature which enable the training of face classifiers to aid in photo tagging (Mitchell, 2011; Breen, 2011). For example, Facebook's tag suggestions feature learns from manually tagged faces to train face classifiers which it can then use to automatically suggest tags on unlabeled images. Tag suggestions are presented to the end-user (in groups of similar photos or by highlighting faces identified as the same person in a variety of photos), for verification before they are accepted as actual tags. Unlike these systems that only allow training of face recognizers, CueFlik enables end-users to train a wide variety of visual concepts.

Related work in the research literature includes Chen *et al*'s system (1999) for interactive clustering and browsing of images. This system is analogous to the Scatter/Gather system for interactive browsing of large document collections (e.g., Hearst *et al*, 1995; Cutting *et al*, 1992). In Chen *et al*'s system, the system automatically clusters images based on both image and text-based features and then presents those clusters to the end-user for perusal. As a person selects clusters of interest, the system will iteratively re-cluster the images. This process continues until images of interest are found by the user. In contrast, CueFlik is intended for the creation of reusable image classifiers rather than as a search and browsing tool (although, it could potentially be used for search as well).

Minka and Picard's FourEyes system (1996) is more similar to CueFlik in that it enables iterative training of image classifiers. As with Chen *et al*'s system, FourEyes presents end-users with automatically generated clusters of images based on pre-defined combinations of features and then the user selects clusters of interest. However, unlike in Chen *et al*'s system, user feedback in FourEyes is used for feature selection and re-weighting in a classifier rather than for selecting data that will then be re-clustered. The FourEyes classifiers can then be used for automatically annotating images in a database. CueFlik differs from FourEyes in that it learns feature weights for image classifiers entirely from user interaction with its machine learning system, as opposed to requiring experts to pre-define feature combinations which FourEyes uses as a basis for feature selection and re-weighting. Furthermore, our work with CueFlik examines a variety of different techniques for displaying system feedback and obtaining end-user guidance during the interactive machine learning process.

4.2 CueFlik

CueFlik (Figure 4.1) enables end-users to interactively train visual concepts for use as reusable image classifiers. End-users train CueFlik by iteratively providing examples of images with and without the desired visual characteristics. During the iterative training process, CueFlik presents examples illustrating its current understanding of the desired concept and end-users use this presentation to decide how to further improve the system's understanding.

Each CueFlik concept is defined as a nearest-neighbor classifier. During the training process, CueFlik uses its current version of this classifier to re-rank a set of images according to their likelihood of membership in the positive class, defined as the distance to the nearest positive example divided by the sum of the distance to the nearest positive and negative example.

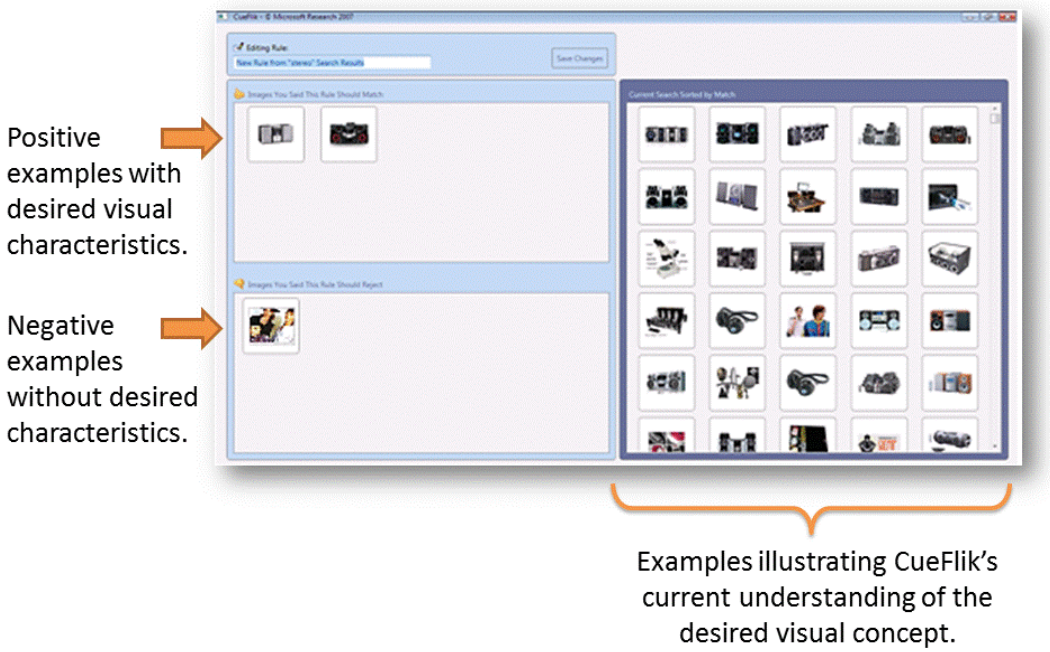

Figure 4.1: CueFlik is an end-user interactive machine learning system that enables end-users to train visual concepts for use as reusable image classifiers. This figure shows CueFlik's interface while an end-user trains a *product photo* visual concept. End-users train CueFlik by iteratively providing example images with and without the desired visual characteristics (left). CueFlik also presents examples illustrating its current understanding of the desired concept (right) and end-users use this presentation to decide how to proceed in training.

Note that CueFlik's nearest-neighbor classifier requires a method for determining the distance between two images. In most nearest-neighbor applications, this distance metric is carefully tuned by an application developer based on their knowledge of the problem. In our case, however, we cannot know what notion of similarity will be appropriate for an end-user's desired concept. CueFlik's concept learning is therefore formulated as a metric learning problem, wherein the system uses the provided examples to learn a distance metric as a weighted sum of component distance metrics. CueFlik currently includes component metrics based on histograms of pixel hue, saturation, and luminosity, an edge histogram, a global shape histogram, and a texture histogram. Formally, CueFlik minimizes the function:

$$f(weights) = \sum_{i,j \in Pos} D(i,j) + \sum_{i,j \in Neg} D(i,j) + \sum_{i \in All} \ln \sum_{j \in All} e^{-D(i,j)}$$

where $D(i, j)$ is the distance metric computed as a weighted sum of CueFlik's component metrics. The first two terms correspond to within-class distances and favor minimizing distance between examples of the same class. The third considers all examples and favors maximum separation. The combination thus

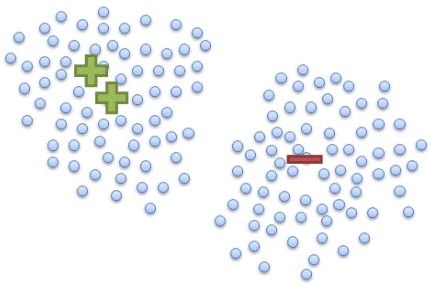

Figure 4.2: CueFlik learns a classifier by finding a distance metric that collapses positive and negative examples together while pushing apart the two classes. The learned distance metric is then used to rank unlabeled images by their proximity to positive examples.

favors weights collapsing each class while maximizing distance between classes. Figure 4.2 illustrates a hypothetical space learned from two positive examples and one negative example.

4.3 Design Factors Affecting End-User Interaction with CueFlik

An end-user's intended goal in interacting with CueFlik is to train a reusable classifier (i.e., the *Intended Product* of interaction is a *reusable* model). Therefore, we assume they will be willing to dedicate time and effort to training CueFlik to recognize an intended visual concept (i.e., interactive training is their primary *Interaction Focus*). However, we also aim to minimize the overall time required to accurately train CueFlik as end-users will likely get frustrated if the system takes too long to understand the end-user's intended concept.

We expect that end-users will have a relatively clear visual concept in mind before their interaction with CueFlik begins (i.e., *Concept Flexibility* will be fixed). This visual concept is what they aim to train CueFlik to recognize. Also, because CueFlik models are intended for reuse, we assume that end-users will aim for accurate recognition of a concept (i.e., accuracy *Performance Requirements* are important). Note that this is a somewhat subjective factor which depends on where CueFlik models will actually be used (e.g., for low risk filtering in search engines versus object detection in camera-based applications). However, for our purposes we directed end-users to aim for accuracy during our evaluations of CueFlik in efforts to simulate these assumed *Performance Requirements*.

We also assume that an end-user will interact with CueFlik in a desktop or laptop environment where they will have a relatively large *Display Size* available for communication with the system.

CueFlik operates over images represented by purely image-based features (i.e., the *Nature of the Data*). Images are straightforward to visualize directly, however the individual features of an image are not (e.g., histograms of pixel hue and edge histograms). CueFlik is also designed to operate over a relatively small

and fixed set of images (i.e, the *Data Source* is static and *Volume* is small), on the order of 1000 images in our experiments. While this set is relatively small (in the big data sense), it still large enough to impact system feedback (as inspecting a set of images this size would still be tedious and time-consuming).

CueFlik is intended for any end-user (i.e., *Individual Differences* should not play a role in how successful an end-user will be in training). In addition, end-users are not expected to have any *Machine Learning Experience* prior to interacting with CueFlik. End-users are also not expected to be familiar with the data set (i.e., they will likely not have previously seen any images over which CueFlik operates). However, because images are simple to inspect in relation to a person's intended visual concept, they could be considered *Domain Experts*.

Some design factors outlined in Chapter 3 were not applicable for our explorations with CueFlik. For example, because our explorations with CueFlik focused on end-user ability to train accurate and reusable models, we did not explore the *Evolutionary Needs* design factor. However, it should be noted that an end-user could continually adapt a trained model within the CueFlik environment. Similarly, while CueFlik classifiers could potentially be shared once created, *Model Ownership* was not the focus of our explorations.

4.4 Illustrating the System's Understanding While Soliciting Effective Training Examples

A fundamental issue in end-user interactive machine learning is illustrating the system's current understanding of a learned concept (i.e., *System Feedback* as discussed in Section 3.2.1). An effective illustration can help people assess the quality of the current concept and in turn inform whether and how they proceed in training. When the actual data over which a system operates is used to illustrate the system's understanding (e.g., via some organization of *Output* examples or predictions, see *Output Type* of *System Feedback* in Section 3.2.1.), then that data can also be used as a source for future training examples (i.e., for *End-User Control* as discussed in Section 3.2.2). When this is the case, it is important that the data presented both effectively illustrates the system's current understanding and also provides the user with informative examples to choose from in further guiding the system.

Prior work with CueFlik compared two methods of illustrating the current version of a learned visual concept using *Output* predictions from the machine learning: the *standard* and the *best and worst matches* method (Fogarty *et al*, 2008). The *standard* method (originally termed the *single* panel method) provided access to the entire set of images, ranked by their likelihood of membership in the positive class (left in Figure 4.3). This is analogous to the traditional information retrieval technique of presenting a ranked list

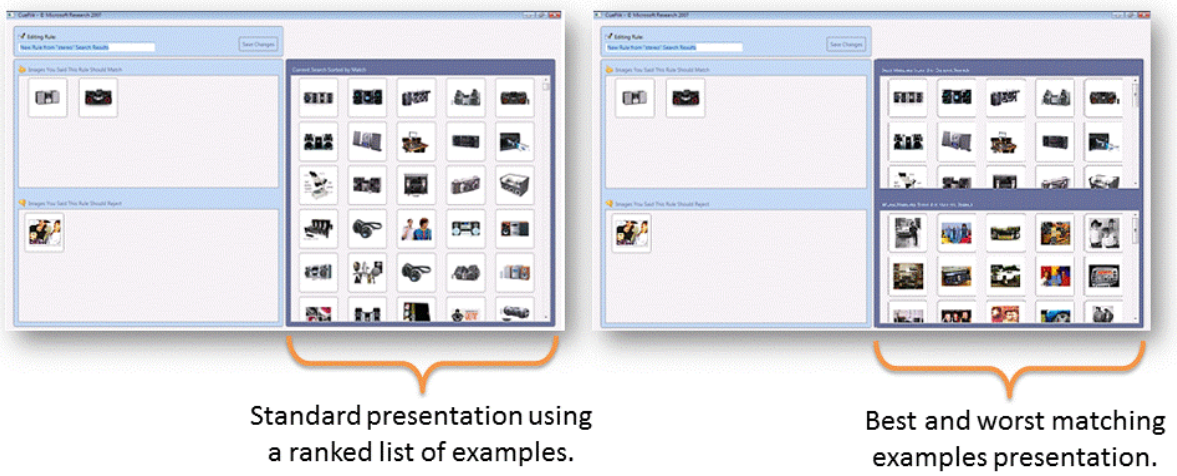

Figure 4.3: Prior work with CueFlik compared two methods of illustrating the current version of a learned visual concept. The *standard* method (left) presented examples ranked by their likelihood of membership to the positive class. The *best and worst matches* method (right) instead showed only a small set of the best and worst matching examples (i.e., examples predicted as positive or negative with high-certainty by CueFlik). An evaluation in prior work showed that the *best and worst matches* technique led end-users to train better models than the *standard* presentation.

of all of the data from which a person can select examples to label for relevance feedback. The *best and worst matches* method (originally termed the *split* panel method) instead showed only a small set of images split across two panels, a best match panel and a worst match panel (right in Figure 4.3). The best match panel showed a small number of high-certainty positive images (images extremely near positive training examples). The worst match panel showed a small number of high-certainty negative images (images extremely near negative training examples).

A formal evaluation with real end-users found that participants using the *best and worst matches* presentation created CueFlik concepts of significantly higher quality, using significantly fewer training examples, and in significantly less time than participants using the *standard* method. One explanation for this improvement over the *standard* method is that *best and worst matches* better convey CueFlik's current understanding by illustrating both the positive and negative regions of the space defined by the currently learned concept (as opposed to focusing on the positive region in a rank ordered list).

However, best and worst matching examples are also extremely similar to already labeled training examples (as shown on the left in Figure 4.4). Therefore, they actually poorly illustrate the range of examples being classified as either positive or negative (right in Figure 4.4). In addition, end-user provided labels on best and worst matches provide the learning algorithm with little additional

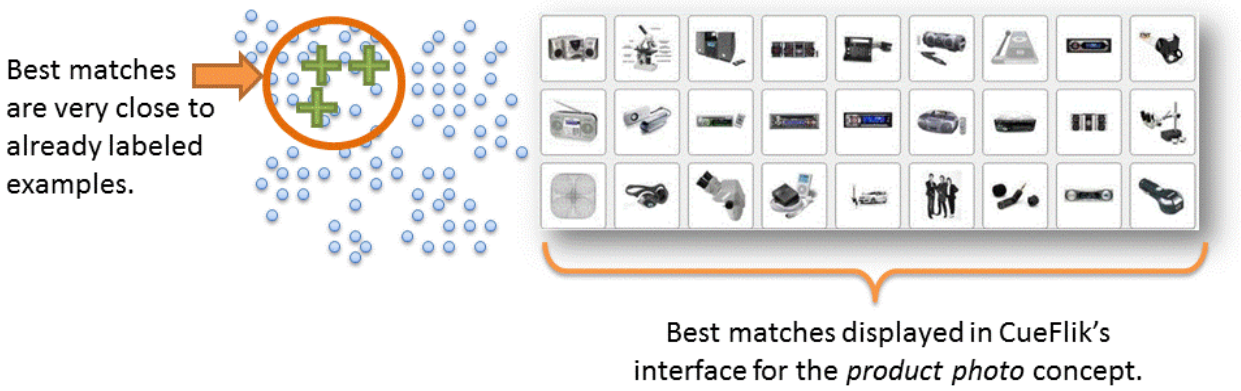

Figure 4.4: The *best and worst matches* technique for illustrating the currently learned concept presents an end-user with examples very close to already labeled training examples (left). Therefore, this technique poorly illustrates the range of examples being classified as either positive or negative by CueFlik's currently learned model (as can be seen on the right in the figure above).

information during the interactive training process because the system is already highly certain that these examples are either positive or negative.

This analysis led us to explore new strategies for selecting examples that both provide the end-user with representative *overviews* of the positive and negative regions (to better illustrate the system's current understanding) while also providing the machine learning algorithm with high-value examples during training. Therefore, in this exploration, we specifically examined new strategies for organizing images for the purposes of *System Feedback* (i.e., we experimented with several strategies for *System Feedback* on *Output*) as well as for guiding the end-user to select informative training examples (i.e., to help provide *Guidance* over *End-User Control*).

In Section 4.4.1 we describe our new overview-based strategies and explain details about how we compute them. Then, in Sections 4.4.2 to 4.4.4, we present our evaluation of these strategies with real end-users and discuss the implications of our results.

4.4.1 Overview-Based Example Selection

Semi-Supervised Classification

Before we can compute overviews of the positive and negative regions of the space defined by a learned concept, we must first detect these regions. That is, we must separate the unlabeled instances into positive and negative regions. A naïve approach would be to apply a nearest neighbor classifier with the current learned distance metric. The limitations of this can be seen in Figure 4.2, where some of the unlabeled instances are clearly in the negative cluster but are also closer to a labeled positive example than they are to any labeled negative example (those on the edge nearest the middle). Because we are focused on

providing overviews of the positive and negative regions, it is important we preserve such relationships among unlabeled instances.

Therefore, we detect the positive and negative regions by employing a semi-supervised classification method that exploits the distribution of both labeled and unlabeled data (Zhu, 2005). Motivated by graph-based methods, we extend the nearest neighbor method to use the underlying data distribution. We create a *k*-nearest-neighbor graph in which each example (i.e., each image) is a vertex and edges are created for the *k* nearest neighbors of each example (we informally found good performance for many values of *k*, and arbitrarily set *k* to 10 in our evaluation). The weight of each such edge is the distance between the corresponding instances according to the current learned distance metric. We then use the graph to compute geodesic distance, the shortest weighted path in the *k*-nearest-neighbor graph, between labeled instances and each unlabeled instance. The positive region is then defined as all instances with a geodesic distance to a labeled positive training example that is less than their geodesic distance to a labeled negative example, and all other instances are included in the negative region. Because geodesic distance considers both labeled and unlabeled data, the resulting separation represents the underlying distribution more reliably than approaches that consider only labeled training examples.

Global Overview Method

Our first strategy for illustrating the system's current understanding while soliciting high-value training examples presents a *global overview*, selecting examples that provide good coverage of the positive and negative regions. Figure 4.5 presents the intuition behind this, showing orange crosses illustrating ten instances selected to provide a *global* overview of the positive region. The same process is applied

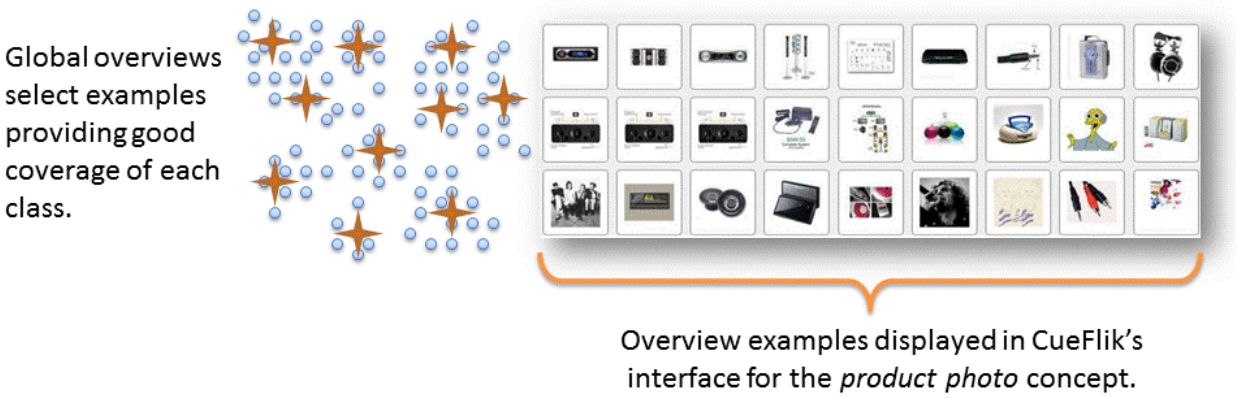

Figure 4.5 Our *global overview* technique for illustrating the currently learned concept selects examples that provide good coverage of the positive and negative classes (see selected examples marked by orange crosses on the left above). These examples are then presented to the end-user in CueFlik (right) to more accurately illustrate the range of examples being classified in each region.

separately to create overviews for the positive and negative regions.

More specifically, we approach the problem of creating a *global* overview as a sensor-placement problem (Krause *et al*, 2008). Each example can be thought of as a possible sensor location where a probe could be placed to sense how well the concept being trained fits the surrounding space. Given a sensor budget (the number of examples we want to use to provide an overview), we choose a set that provides maximum information about the concept the end-user is attempting to train. Intuitively, our approach builds upon the observation that instances that are close to each other can be expected to share similar properties; thus our selection scheme utilizes the principle of maximizing the information theoretic criteria of mutual information. In particular, our aim is to select a subset of instances that share the highest mutual information with the rest of the high-dimensional space and is therefore most representative of the full set. However, selecting such a set is NP-complete, and a greedy and myopic selection procedure maximizing mutual information gain is the best possible approximation (Krause *et al*, 2008). We therefore employ this approach.

Formally, we consider a Gaussian Process perspective with covariance function (alternatively, kernel function):

$$k_{ij} = e^{\frac{-G(i,j)^2}{\text{Mean}(G^2)}}$$

where *G(i, j)* is geodesic distance in a *k*-nearest-neighbor graph based on the current learned distance metric, and G^2 is the square of each element in *G*. Because of the negative exponent, k_{ij} measures similarity between instances *i* and *j* and ranges from 0 (infinitely far apart) to 1 (at the same location). The matrix *K* therefore encodes similarity and how well information flows in the space. Including *i* in an overview provides a good representation of instances where k_{ij} is high, but is unrepresentative of the portion of the space where k_{ij} approaches zero.

At each step in the greedy selection, given the existing set of selected instances *S* and unselected instances *U*, we select the instance that maximizes the gain in mutual information on the remainder of the unselected instances:

$$MI(U-i; S \cup i) - MI(U-i; S)$$

For Gaussian Process models, this can be achieved by selecting the instance *i* that maximizes:

$$f(i) = \frac{1 - K_{i,S} K_{S,S}^{-1} K_{S,i}}{1 - K_{i,U-i} K_{U-i,U-i}^{-1} K_{U-i,i}}$$

where $K_{S,S}$ is the similarity matrix among S, $K_{U\text{-}i,U\text{-}i}$ is the similarity matrix among U excluding i, and $K_{i,S}$, $K_{S,i}$, $K_{i,U\text{-}i}$, and $K_{U\text{-}i,i}$ are each similarity vectors between i and the respective sets. The numerator characterizes the similarity of i to the previously selected examples and the denominator characterizes the similarity of i to other currently unselected examples. Choosing i that maximizes the ratio selects an example that is farthest from previously selected examples (most dissimilar to those previously selected) but also central to the unselected examples (most representative of those currently unselected).

Projected Overview Method

Our second strategy presents instances selected after they have been *projected* onto a set of principal dimensions corresponding to the major axes of variation in a dataset. Figure 4.6 illustrates this, showing two principal dimensions with five examples selected for each. This strategy was intended to convey variation along feature dimensions being classified as either positive or negative (i.e., *System Feedback* is on *Features* of the data).

The approach is similar to Principal Component Analysis or Multi-Dimensional Scaling, except that we want a method that corresponds to the structure of the underlying data (for much the same reasons that we motivated semi-supervised classification). We therefore apply an ISOMAP-based non-linear projection technique (Tenenbaum *et al*, 2000), which selects principle dimensions using geodesic distance in a *k*-nearest-neighbor graph. The process of selecting principal dimensions is identical to that used in Multi-Dimensional Scaling, except using geodesic distance.

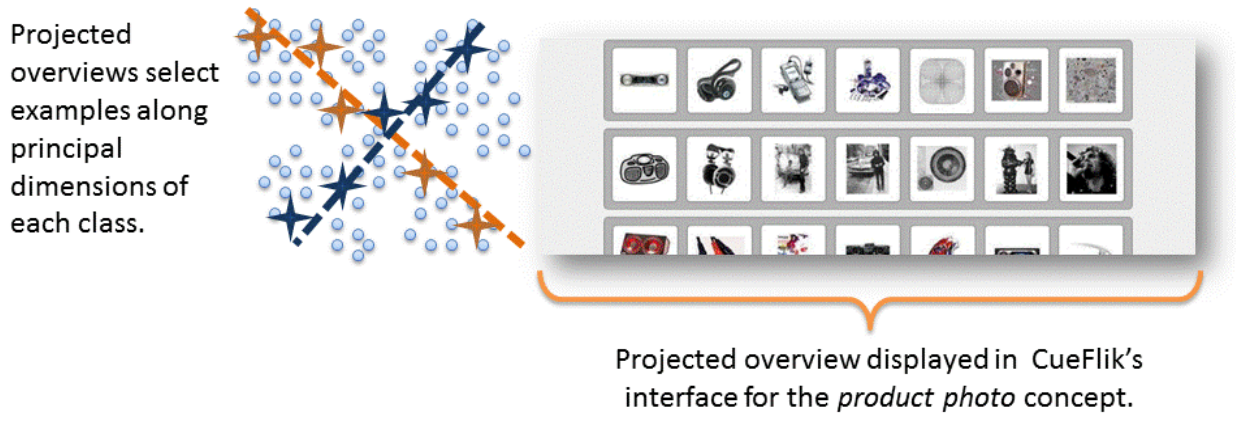

Figure 4.6: Our *projected overview* technique for illustrating the currently learned concept selects examples along principal dimensions of each class (left). Examples selected along each dimension are displayed in rows in CueFlik's interface (right). This technique is designed to convey variation along principal dimensions of a region.

After obtaining a set of principal dimensions, it might seem that we could apply the same sensor-placement strategy from the previous section to each one-dimensional space to choose instances that illustrate that principal dimension. The problem with such an approach, however, is that it leaves selection completely unconstrained in the other dimensions. Although we are trying to select image instances to illustrate variation along a single principal dimension, examples selected in this way may do a poor job of conveying the intended variation because they may also vary in many other dimensions. We therefore derive a new scheme to select examples that provide coverage of a single principle dimension but also vary as little as possible in all of the other principal dimensions. We find *i* maximizing:

$$f(i) = \left(\frac{1 - K_{i,S} K_{S,S}^{-1} K_{S,i}}{1 - K_{i,U-i} K_{U-i,U-i}^{-1} K_{U-i,i}} \right) \left(\frac{1}{1 - \overline{K}_{i,S} \overline{K}_{SS}^{-1} \overline{K}_{S,i}} \right)$$

where K is the similarity matrix for the principal dimension for which we are currently selecting a set of representative examples and $\overline{K}$ is the similarity matrix for all of the other principal dimensions. The left term again prefers examples farthest from previously selected examples but also central to unselected examples in the current principal dimension. The right term prefers selected examples that are similar to each other in all other dimensions. Maximizing the product selects examples that span a particular dimension and are similar to each other in all other dimensions. In Figure 4.6, this additional term is responsible for the roughly collinear set of selected instances for each principal dimension.

Using Neighbors to Illustrate Commonalities

While our *global* and *projected* overviews present representative examples from both the positive and negative regions, they do not convey a notion of *why* the system may be classifying any given image as positive or negative. Therefore, to help illustrate the image characteristics influencing the system's classifications, we experimented with pairing both our global and projected overview examples with their nearest *neighbors* (see Figure 4.7). Here, our intuition was that a group of similar (i.e., nearby) instances would help an end-user recognize their common characteristics and better interpret an overview (e.g., a group of images with a similar hue but varying in shape would indicate the system is being influenced by that particular hue). This strategy therefore conveys aspects of CueFlik's underlying machine learning (i.e., *System Feedback* is on *Logic*). This strategy also encourages labeling entire groups of similar images as positive or negative in a single action (i.e., course-grained *Control Granularity*) for potentially faster training.

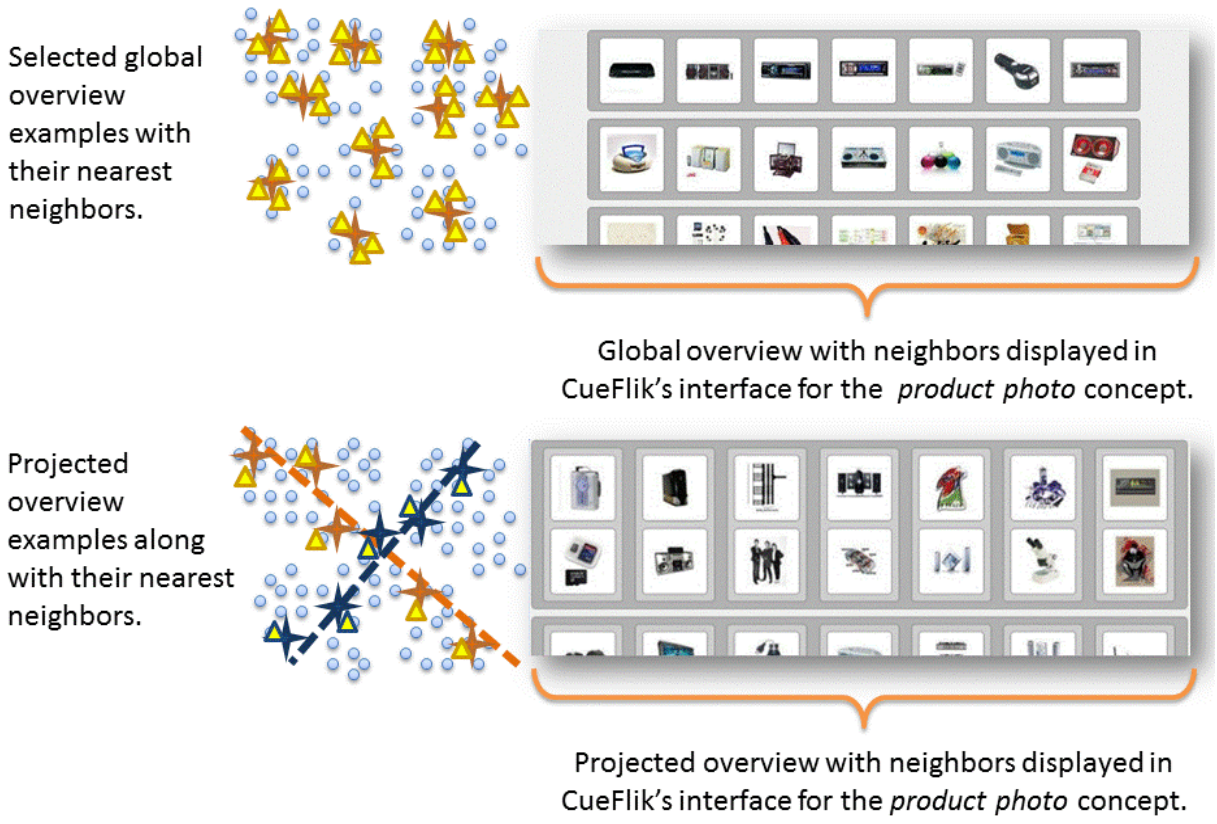

Figure 4.7: We experimented with pairing both our *global* and *projected overview* examples with their nearest *neighbors* (neighbor examples are shown as yellow triangles on the left above). For the *global overview with neighbors* technique (top right), CueFlik displays groups of examples (i.e., a selected global overview example along with its nearest neighbors) as rows in its interface. For the *projected overview with neighbors* technique (bottom right), CueFlik displays examples selected along each projection as rows as before, and displays groups (i.e., a selected example along with its single nearest neighbor) as columns along each row. *Neighbors* are designed to better convey why a selected example is being classified as positive or negative (by illustrating common visual characteristics in a group).

Downsampling for Performance

Both overviews require inversion of similarity matrix K, an expensive operation. To maintain interactive speeds during the interactive machine learning process, we randomly downsample the number of instances in the positive and negative regions before creating overviews. The success of overviews in our evaluation shows this is not problematic, but it is likely important that this be random to preserve the original distribution. Parallel matrix inversion algorithms can leverage the increasing number of cores in a typical computer, and the degree of downsampling could be automatically adjusted to maintain responsiveness on a particular computer. Our evaluation fixed downsampling to 200 images in each of the positive and negative regions. In conditions that display neighbors, the neighbors are taken from the original full set of images.

4.4.2 Evaluation

We conducted a within-subjects experiment to evaluate our new *overview*-based strategies and compare them to the best performing strategy from previous work (i.e., the *best and worst matches* strategy presenting examples that the system is highly certain are either positive or negative). The experiment was a 2 (*Global* vs. *Projected*) × 2 (*NoNeighbors* vs. *WithNeighbors*) design with an additional *BestAndWorstMatches* baseline condition.

Interface Conditions

We tested a total of five interfaces in our experiment, intentionally holding constant the number of images displayed so results were not confounded by this factor:

- *Baseline/ BestAndWorstMatches*. Presented the 42 highest-certainty images from both the positive and negative regions (see Figure 4.4). This strategy yielded the best results in prior work (Fogarty *et al*, 2008).

- *GlobalNoNeighbors*. Selected 42 images from both the positive and negative regions using our global overview method (see Figure 4.5).

- *GlobalWithNeighbors*. Selected 6 images from both the positive and negative regions using our global overview method, and presented each of them together with 6 neighbors (see top of Figure 4.7).

- *ProjectedNoNeighbors*. Projected both the positive and negative regions onto 6 principal dimensions accounting for the greatest variance, and selected 7 images to illustrate the variation in each principle dimension (see Figure 4.6).

- *ProjectedWithNeighbors*. Projected both the positive and negative regions onto 3 principal dimensions accounting for the greatest variance, selected 7 images to illustrate the variation in each principle dimension, and presented each together with a single neighbor (see bottom of Figure 4.7).

The *BestAndWorstMatches* and *Global* interfaces sorted images (and groups of images) in order of their certainty. The *Projected* interfaces sorted principal dimensions by the amount of variance explained by the dimension and then sorted images within each dimension into a spectrum.

Tasks

Participants trained two concepts per interface condition, using queries and corresponding concepts developed for the previous CueFlik evaluation (Fogarty *et al*, 2008). Each task consisted of approximately

1000 images corresponding to a query (e.g., “drink”, “sky”) and a set of ten target images printed on a sheet of paper and labeled with a target concept (e.g., “pictures with products on a white background”, “pictures with quiet scenery”). Participants were asked to create a concept such that the target images would be highly ranked if they were actually contained in the image set. Target images were originally well distributed across the ranking of the result set so that improvement in their ranking could be used as a measure of the quality of the trained concept. Importantly, the target images were filtered from the result set so that participants could not see how their current learned concept ranked the targets.

Study Design and Procedure

We ran participants in pairs, with each working on an identical 3.16 GHz quad-core Dell T5400, with 8 GB of RAM and 512 MB Video, running Windows XP, and using 24” Dell monitors at a resolution of 1920×1200.

Before beginning, the experimenter demonstrated how to use CueFlik to build a simple rule favoring images with a vertical aspect ratio. We used the *GlobalWithNeighbors* condition for the demonstration so that we could explain that some interfaces would group images and that participants could guide the system by selecting individual images, ctrl-selecting multiple images, or selecting grouped images. Participants then practiced training a simple concept (images of maps within a query for “Seattle”) using the first interface condition that they were to interact with during the main portion of the study.

Because we did not expect queries would lead directly to carryover effects, and because we wanted to ensure balanced coupling of interface conditions with queries, we fixed the order of queries in the experiment. We manually categorized target concepts as easy or difficult and then pseudo-randomly selected queries such that the first task in each interface condition would be an easy concept and the second task would be a difficult concept. The ordered query pairs were: “character” (*clipart*) and “sky” (*quiet scenery*), “drink” (*product photo*) and “bill” (*portrait*), “car” (*product photo*) and “crowded” (*cluttered*), “stereo” (*product photo*) and “disco” (*brightly colored*), and “tennis” (*clipart*) and “sea” (*quiet scenery*).

The order of interface conditions was counterbalanced using a Latin square design. Each participant trained the pair of easy and difficult tasks before proceeding to the next interface. For each task, participants were given a printout containing the target images labeled with the target concept and clicked a button to begin. This was in order to simulate a defined visual concept that the user would be expected to have in mind when training CueFlik (*Concept Flexibility*).

Participants were told to perform each task as quickly and accurately as possible (to simulate expected accuracy *Performance Requirements* of the interaction process). They could also click on a button in the interface to advance to the next task if they felt their concept was not improving. We also imposed a maximum time limit of 4 minutes (indicated 20 seconds prior by a visual warning), after which the task would self-advance. After each task, a dialog appeared giving the participant's final score on the task (computed as the final mean ranking of the targets in the image set, inverted and converted to a percentage).Participants were then given a new printout and began the next task. All actions were timestamped and logged.

After each interface condition, participants were given a short questionnaire asking for their level of agreement (on a 5-point Likert scale) with five short statements regarding the interface they had just used. The questionnaire also asked the participants to briefly explain what they thought CueFlik had been doing each time they gave it more training examples, asked what they thought would help improve the interface they had just used, and asked for any other comments. A different short questionnaire at the end of the experiment asked for overall aspects of CueFlik that participants liked, disliked, or would like improved.

Participants

Twenty people (ten female, ages 20 to 48) volunteered. None were colorblind, and all had 20/20 or corrected to 20/20 vision. We attempted to recruit a balance of image search novices (performing no more than one image search every week) and image search experts (performing more than five searches weekly), obtaining seven self-reported novices and thirteen self-reported experts. The experiment lasted approximately 90 minutes, and participants were given a software gratuity for their time.

4.4.3 Results

We analyze our results using four dependent measures. Consistent with prior work (Fogarty *et al*, 2008), we define *Score* as the mean ranking of the target images by a learned concept. A lower score indicates a higher-quality concept that ranks targets closer to the top of the query results. We define *Time* starting from the button click that began each task. Because participants could label multiple images per action (e.g., by dragging grouped images or ctrl-selecting multiple images or groups), we define *NumImages* and *NumActions* as the number of images or actions used to train a concept.

We perform all of our analyses using mixed-model analyses of variance. All of our models include *Participant* and *Query* as random effects. Modeling *Participant* accounts for any variation in individual performance, and modeling *Query* accounts for any difference in the difficulty of queries or their associated concepts as well as any learning, fatigue, or other potential carryover effects between tasks

(because our experiment fixed query order). Throughout this section, we report least-squared means obtained from our mixed-model analyses.

BestAndWorstMatches versus Overview

We first compared our baseline *BestAndWorstMatches* condition to all of our *Overview* methods (i.e., aggregating data for our four *Overview* methods). Mixed-model analyses reveal an apparent tradeoff: participants using *Overview* interfaces create concepts with a significantly better final *Score* (178 vs. 223, $F1,170 = 8.86$, $p \approx .003$) but spend significantly more *Time* training their final concept (197 sec vs. 176 sec, $F1,170 = 7.72$, $p \approx .006$) in comparison to the *BestAndWorstMatches* interface. We found no difference in the *NumImages* or *NumActions* for the final concept.

We observed during our experiments that participants often continued providing additional training examples even when they did not seem to be further improving a concept. This obviously adds to the *Time*, *NumImages*, and *NumActions* used to train a final concept, and we believed it could also negatively impact the *Score* of a final concept. We therefore further analyze the point where participants obtained their *Best* learned concept, defined as the first point at which their *Score* for a task was within 5% of the best *Score* they achieved at any point during that task. This metric detects both well-defined peaks as well as the beginning of gradual-sloping plateaus. Figure 4.8 plots this perspective, showing the average *BestScore* and *BestTime* for each interface and their corresponding *FinalScore* and *FinalTime*. For the sake of readability, Figure 4.8 plots *Score Improvement*, obtained by subtracting *Score* at each plot point from the initial *Score*. We now analyze these and the *Decay* from *Best* to *Final* (the time, images, and actions spent continuing to train a model after achieving *BestScore* as well as the resulting negative impact on *FinalScore*).

Mixed-model analyses show that *BestScore* is significantly better for *Overview* interfaces than for *BestAndWorstMatches* (128 vs. 149, $F1,170 = 6.52$, $p \approx .012$) and that participants reach their *BestScore* in the same amount of *Time* (108 sec vs. 120 sec, $F1,170 = 1.11$, $p \approx .294$). We also saw a marginal effect on *NumActions*, with *Overview* interfaces requiring marginally less actions to reach the significantly better *BestScore* (6.39 vs. 8.63, $F1,170 = 3.89$, $p \approx .050$). We found no difference in *NumImages* to *BestScore*.

Examining the *Decay* from *Best* to *Final*, a mixed-model analysis shows *Overview* interfaces result in marginally less *ScoreDecay* than the *BestAndWorstMatches* interface (49 vs.74, $F1,170 = 3.44$, $p \approx .065$), even though the *Overview* interfaces result in both greater *DecayTime* (89 sec vs. 56 sec, $F1,170 = 9.92$, p

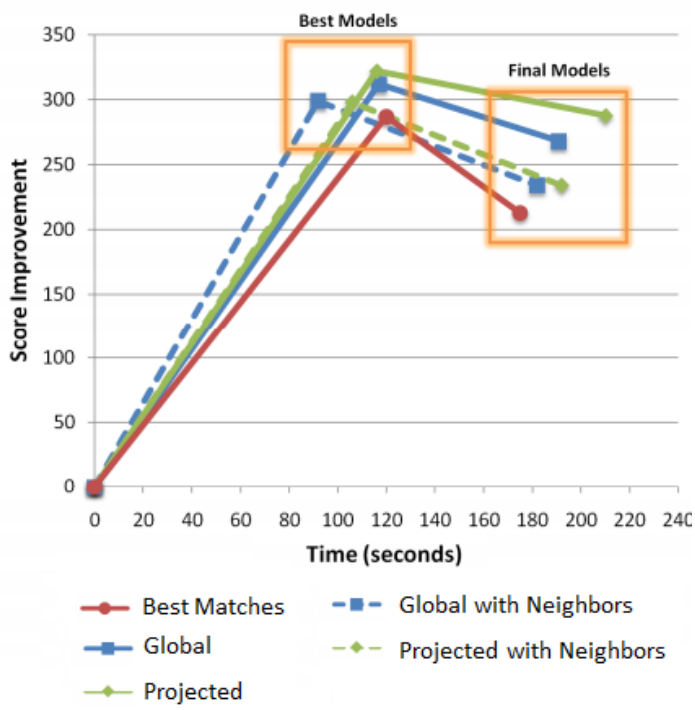

Figure 4.8: Our *Overview* techniques led participants to train better *Best* models with CueFlik than the *Best Matches* technique (the best performing technique from prior work). In addition, our *Overviews* reduced the magnitude of *Decay* on our participants *Final* models.

≈ .002) and *DecayActions* (8.03 vs. 5.30, $F1,170 = 7.49$, $p \approx .007$) than *BestAndWorstMatches*. We found no difference in the number of *DecayImages*.

We defer discussion of freeform feedback, but analyses of our post-condition questionnaire Likert scales found no significant differences for *BestAndWorstMatches* versus *Overview*.

Examining OverviewMethod and Neighbors

To examine differences among our *Overview* interfaces, we exclude data from our baseline *BestAndWorstMatches* condition and conduct a series of mixed-model analyses with fixed effects *OverviewMethod* (*Global* vs. *Projected)*, *Neighbors* (*NoNeighbors* vs. *WithNeighbors*), and their interaction *OverviewMethod* × *Neighbors*. In all cases, effects include either *OverviewMethod* or *Neighbors*, but not both and not their interaction. We therefore only report the effecting treatment (i.e., removing the non-effecting treatment and the interaction from each analysis).

We first examined our *Final* measures, finding interfaces with *NoNeighbors* yielding a significantly better *Final Score* than interfaces *WithNeighbors* (159 vs. 197, $F1,131 = 10.7$, $p \approx .001$). We also find that *Global* overviews led participants to spend significantly less *Time* training a *Final* concept than *Projected*

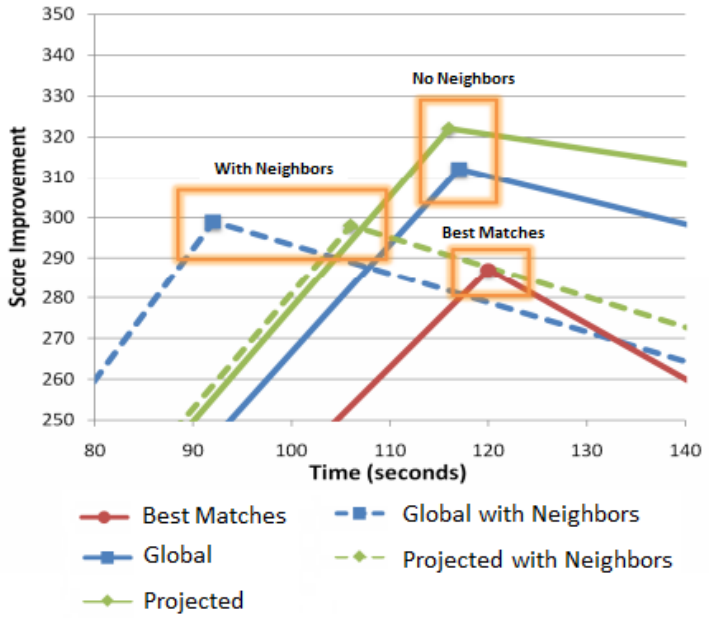

Figure 4.9: Examining differences among our *Overview* interfaces. *NoNeighbors* interfaces led participants to select training examples that result in significantly better *Best*, while *WithNeighbors* interfaces result in marginally faster model training.

overviews (187 sec vs. 206 sec, $F1,130 = 10.1$, $p \approx .002$). There were no significant effects on *NumImages* or *NumActions*.

Examining the *Best* concepts shows that interfaces with *NoNeighbors* result in a significantly better *Best Score* than *WithNeighbors* (119 vs. 138, $F1,130 = 9.34$, $p \approx .003$). We also find that participants require marginally less time to reach their *BestScore* for interfaces *WithNeighbors* than with *NoNeighbors* (99 sec vs. 117 sec, $F1,130 = 3.21$, $p \approx .076$). There were no significant effects on *NumImages* or *NumActions*. This pair of *Neighbors* results is shown in Figure 4.9, a close-up of the *Best* region from Figure 4.8. The finding that *NoNeighbors* interfaces yield a better *BestScore* than *WithNeighbors*, together with the finding that *Overview* interfaces yield a better *BestScore* than *BestAndWorstMatches*, implies a transitive relationship between *NoNeighbor* interfaces and *BestAndWorstMatches*. A mixed-model analysis of variance confirms *NoNeighbors* overviews result in a significantly better *BestScore* than *BestAndWorstMatches* (119 vs. 149, $F1,91 = 10.67$, $p \approx .002$). Similarly, a mixed-model analysis of variance shows *WithNeighbors* overviews require marginally less *Time* to reach a *BestScore* that is as good as the *BestAndWorstMatches BestScore* (99 sec vs. 120 sec, $F1,91 = 2.97$, $p \approx .088$).

Examining the *Decay* from *Best* to *Final*, we find only that interfaces *WithNeighbors* resulted in marginally more *DecayActions* than interfaces with *NoNeighbors* (8.79 vs. 7.28, $F1,129 = 2.91$, $p \approx .091$). There were no significant effects on *ScoreDecay*, *DecayTime*, or *DecayImages*.

Analyses of our post-condition questionnaire Likert scales showed marginal effects of *Neighbors*. Participants agreed marginally more with "*I felt confused or frustrated when trying to create rules*" for the *WithNeighbors* interfaces than the *NoNeighbors* interfaces (2.5 vs. 2.1, $F1,1 = 3.03$, $p \approx .086$) and marginally less with "*I felt that this method was easy to use*" (3.8 vs. 4.1, $F1,1 = 2.80$, $p \approx .098$).

4.4.4 Discussion and Future Work

Our experimental results substantiate our hypothesis that providing an *overview* of positive and negative regions leads end-users to select better training examples when interacting with CueFlik. Using the same amount of *Time*, *NumImages*, and *NumActions*, participants using *Overview* interfaces trained *Best* concepts that were significantly better than those trained using the *BestAndWorstMatches* interface, the best-performing interface from prior work (Fogarty *et al*, 2008). Notably, this prior work shows that the split *BestAndWorstMatches* approach performs significantly better than presenting the full image set and found no effect of presenting uncertain images identified using active learning heuristics (Fogarty *et al*, 2008). Furthermore, although participants using *Overview* interfaces spent significantly more *Time* and *NumActions* in the *Decay* from *Best* to *Final*, their *ScoreDecay* was still less than that of the *BestAndWorstMatches* interface. This lower rate of *Decay* per unit of *Time* and *Actions* provides further evidence that our *Overview* interfaces led participants to select better training examples (i.e., *Overviews* effectively *Guided End-User Control)*.

Our *WithNeighbors* overviews were intended to provide context in which end-users could better interpret why the system was classifying images as either positive or negative, as well as the possibility of quickly training a concept by labeling entire groups of similar images in one step. *WithNeighbors* overviews did require marginally less *Time* to obtain a *BestScore* as good as that for *BestAndWorstMatches*, but *NoNeighbor* interfaces had significantly better *Scores* for both their *Best* and *Final* concepts. A potential commonality here might be that the *BestAndWorstMatches* and *WithNeighbors* interfaces are less effective because they less effectively convey variation with a set of images. In addition to our performance analyses, this interpretation is supported by negative responses to *WithNeighbors* on our questionnaire scales and by participant feedback for these interfaces of the tone "*there were nowhere near enough images – too many were duplicated*" and "*give more options of photos so filtering them will go faster/easier*". Showing variation or diversity has proven to be an effective feedback organization strategy in relevance feedback and recommender systems (see Section 3.2.1) as it enables end-user to more

quickly navigate to desired items and avoid undesired items. Our results suggest that variation is also useful for enabling end-users to find appropriate items for training a reusable model.

Interestingly, the same emphasis on variation that leads end-users to choose better examples may also make it more difficult to assess how well the current version of a learned concept is performing (because the overview provides more insight into less central portions of a positive or negative region). This potentially suggests a separation of *System Feedback* from *End-User Control* mechanisms, however, further investigation into this hypothesis is necessary.

All of the interfaces suffered from some *Decay*. Participant feedback included many comments of the form "*it was weird, sometimes it would start out doing really well, but as I kept going it did worse*" and "*it is hard to know if more data is better as I should probably stop occasionally to see the results as I am going*". However, our *Overview* interfaces marginally reduced the impact of that decay on concept *Score*, but also had greater *DecayTime* and *DecayActions*. This problem led us to our second exploration with CueFlik examining the potential for end-user exploration and revision of the model space as will be discussed next.

4.5 Examining Multiple Potential Models

During our first exploration with CueFlik, we observed that participants sometimes continued providing training examples even when they did not seem to be further improving a concept. Participant feedback indicated that they often inadvertently directed CueFlik towards undesired behaviors (e.g., "*it was weird, sometimes it would start out doing really well, but as I kept going it did worse*"). We hypothesized that this was partly due to an implicit assumption in prior research about how people should interact with machine learning. Machine learning systems learn by generalizing from examples of classes of objects. Prior research has thus focused interaction on prompting a person to answer "*what class is this object?*" (e.g., Tong and Chang, 2001; MacKay, 1992). Such an approach permits simulated experiments with fully-labeled datasets. However, treating a person simply as an oracle neglects human ability to revise and experiment. We therefore hypothesized that a more effective approach may be asking a person to consider "*how will different labels for these objects impact the system in relation to my goals?*"

Consider a person attempting to use CueFlik to train a "portrait" concept, as in Figure 4.10. Given a large and diverse set of images, the person may label images that are clearly portraits as positive and images that are clearly not portraits (e.g., line drawings) as negative. These are unambiguous given the person's goals. However, they may also encounter images that are more ambiguous, such as a close up of a person's face on a magazine cover (middle of Figure 4.10). This image features a prominent face, but also

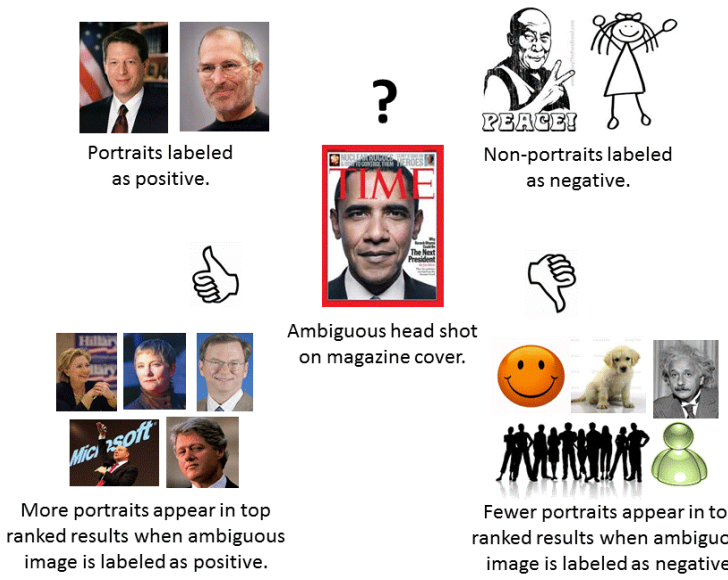

Figure 4.10. Our second exploration with CueFlik examined end-user comparison of multiple potential models based on the insight that whether or not this magazine cover is a *portrait* may be less important than which resulting model is preferable.

a variety of surrounding graphics the person does not intend as part of the "portrait" concept. We propose that a person should be able to experiment with either potential labeling (i.e., positive or negative), compare the resulting concepts, and then decide upon a label that guides the system to learn the desired concept.

Comparison of alternatives is a proven technique in human-computer interaction (e.g., Norman, 1988), but has not been explored in the context of people interacting with machine learning. Deleting training examples, whether explicitly, by undo, or by rolling back to a previous model, is a poor fit from the machine learning perspective because it deletes information that has already been made available to the machine learning system. General machine learning tools support comparison of different model parameterizations for a given set of data (e.g., Weka, see Witten and Frank, 2005), but do not support comparing models as they evolve during interactive training. Furthermore, such tools target people skilled in statistical machine learning rather than the general end-user.

Therefore, in our second investigation with CueFlik, we examined the impact of end-user comparison of multiple potential models during the interactive machine learning process. This exploration therefore examines *Temporal* aspects of the interactive machine learning process. Specifically, we examine techniques for realizing both the forward and backward *Control Direction* (see Section 3.2.3).

In Section 4.5.1, we present our techniques for supporting model comparison and exploration during the interactive machine learning process. Then, in Section 4.5.2 to 4.5 we present our evaluation, results and discussion of these techniques.

4.5.1 Supporting Model Comparison and Revision

To enable model exploration, we augmented CueFlik with a history visualization showing recently explored models and support for revision. The history visualization (bottom, Figure 4.11) contains a plot of each model's estimated quality, updated after every end-user interaction (e.g., labeling examples). Because there is no ground truth model (this is exactly what the system is trying to learn from the end-user), model quality is estimated using leave-one-out-cross-validation on the current set of training examples (which represent a sample of the desired model behavior), with confidence intervals computed according to a binomial distribution (Kohavi, 1995). The history also shows snapshots of each model's top ranked images for visual comparison. Therefore, the history shows feedback about CueFlik's underlying model at different levels of *Feedback Granularity* (see Section 3.2.1). The history is intended to help people visually compare and assess the relative quality of the models they have trained.

We provide three mechanisms for supporting revision of a person's currently trained model. First, a person can undo or redo actions. Second, a person can remove labels of individual examples if they feel the example could be hurting the model's performance. Finally, a person can click directly on a data point or snapshot in the history visualization to revert back to that model. The person can then continue providing examples from that stage, effectively enabling a simple branching mechanism for exploring multiple models. Conceptually, revision extends the traditional feedback loop by allowing a person to branch off and explore different alternatives as a part of determining how to best obtain their desired concept. That is, revision enables end-user control in both the forward and backward *Control Direction* (see Section 3.2.3).

Both our history visualization and revision capabilities also provide an end-user with a *Variety* of *System Feedback* (see Section 3.2.1) and *End-User Control* (Section 3.2.2) mechanisms during the interactive machine learning process.

Figure 4.11: To enable end-user exploration of multiple potential models during the interactive machine learning process, we augmented CueFlik with a history visualization (bottom of figure) and support for revision. Revision mechanisms include undo and redo (top right), removal of labeled examples (left), and clicking directly on the history visualization to revert back to previous models (bottom of figure).

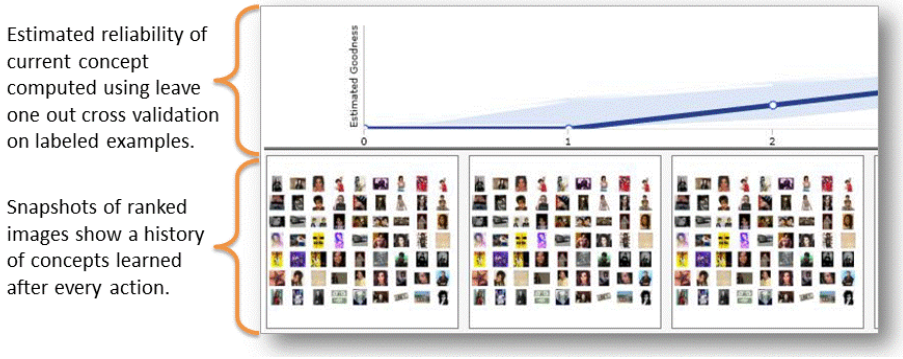

Figure 4.12: Close-up of CueFlik's history visualization. The history visualization includes *snapshots* of the top ranked images taken after every action (bottom) along with a *plot* of each models estimated quality (top). Quality estimates are computed using leave-one-out cross validation on labeled examples.

4.5.2 Evaluation

We conducted an experiment to understand how people would use our history and revision enhancements to examine multiple potential models in CueFlik and to determine the effectiveness of a revision-based approach to interactively training a machine learning system.

We used a 2 (*History* vs. *No History*) × 2 (*Revision* vs. *No Revision*) within-subjects design (with conditions counterbalanced using a Latin Square). We followed a study procedure analogous to our previous exploration (see Section 4.4.2). Again, participants were asked to train the system as accurately and quickly as possible and all actions in CueFlik were timestamped and logged.

Nineteen participants (8 female, ages 18-45) volunteered for thus evaluation (a twentieth was unable to attend as scheduled) and were given a software gratuity for their time.

4.5.3 Results

Analysis of logged data from our experiment shows that participants made use of our revision mechanisms to explore multiple potential models in CueFlik. When the history visualization was available, participants made revisions in 68% of their tasks. To make revisions, participants used the undo/redo feature in 19% of tasks, the remove label feature in 5% of tasks, and the ability to rollback to previous models via the history in 42% of tasks. When the history visualization was not available, participants made revisions in 41% of tasks. Interestingly, their usage of the undo/redo and remove label features increased to 30% and 11%, respectively, likely because these were the only revision mechanisms that were available in this condition.

For the tasks in which people made revisions when the history was available, 3% of their total actions (labeling examples and revision of any kind) were undo/redo actions, 1% were removing labels, and 9% were rolling back via the history. Without history, they relied more on the undo/redo and remove label features, using undo/redo in 11% of actions and removing labels in 3% of actions. This suggests participants were able to make progress by providing CueFlik with examples, but in some cases felt it necessary to explore or revert back to previous models.

To analyze the impact of our revision and history enhancements on participant ability to effectively train a concept, we measured their *Time* to complete each task, the *NumImages* and *NumActions* taken to train their final models, and their final model *Scores*. As in our first evaluation, *Score* is defined as the mean ranking of the target images by the final learned concept, where a lower score indicates a higher-quality concept (i.e., targets are nearer the top of ranked examples). We perform these analyses using mixed-model analyses of variance. All of our models include *History* (*History* vs. *NoHistory*), *Revision* (*Revision* vs. *NoRevision*), and their interaction *History*×*Revision* as fixed effects. To account for any variation in individual performance, query difficulty, or other carryover effects, we include *Participant* and *Query* as random effects.

Our analyses reveal that participants spent more *Time* creating models with *History* available (160 vs. 146 seconds, $F1,118=3.93$, $p = .049$) and performed more *NumActions* with *History* than without (17.2 vs. 15.1, $F1,118 =4.55$, $p = .035$). There was no effect of *History* on *Score*. Participants also performed more *NumActions* when *Revision* capabilities were enabled compared to *NoRevision* (17.2 vs. 15.2, $F1,118=4.08$, $p = .046$). There were no effects of *Revision* on *Time* or *NumImages*. In terms of final *Scores*, we found that participants created better models with *Revision* (211 vs. 242, $F1,119 =3.57$, $p = .061$). There were no interaction effects on any of our dependent measures.

4.5.4 Discussion and Future Work

Our evaluation shows that participants made use of revision mechanisms while interactively training a machine learning system and that this led them to achieve better final models in the same amount of time (even while performing more actions). Furthermore, being able to examine and revise actions is consistent with how people typically expect to interact with their applications. One participant commented that without revision "*it felt a little like typing on a keyboard without a backspace key, like stabbing myself in my eye with a pen*". Therefore, this exploration provides evidence for the benefits of enabling end-users to both direct and revise their interactions with a machine learning system (i.e., interact in both the forwards and backwards *Control Direction*). This aspect of the *Control Direction* design dimension therefore warrants examination in other interactive machine learning-based systems.

In contrast, our history visualization enhancement led participants to spend more time and perform more actions to train concepts without improving overall model quality. While some participants seemed to find the history helpful for examining different models (e.g., "*[the visualization was] helpful to see if I was heading in the right direction*"), observations during the study and other participant comments indicate that the plot was generally distracting (e.g., "*I felt like I was concentrating too much on the line graph.*"). Although the plot used an accepted machine learning metric to estimate model reliability (leave-one out cross validation accuracy), end-users seemed to use it less like an approximation tool for helping them interpret model quality and more like a quantity to maximize (e.g., *"I wanted the graph to go up instead of concentrating on [the ranked results]"*). This underscores the impact of a person's understanding (or lack thereof) of the capabilities and limitations of machine learning systems (i.e., *Machine Learning Expertise*). Future work might therefore examine alternative methods for conveying estimated quality or methods for improving a person's mental model of the capabilities of the system (i.e., examining the *Human Learning* design dimension).

Note that participants did use the history for reverting back to previous models, suggesting that the history may be beneficial as an additional facility for enabling revision (e.g., "*[the history] helped because when*

I started to get off track I could always go back and try a different route"). In addition, the fact that participants used all mechanisms for revision when they were available suggests a benefit of providing multiple methods of end-user control (i.e., *Control Variety*). However, we hypothesize that future designs should consider balancing *Control Variety* with *Guidance* as providing too many mechanisms for control might make it more difficult to select the most effective option.

4.6 Summary

Our explorations with CueFlik examined various design dimensions for supporting end-user interactive machine learning of accurate and reusable models. Because interactive machine learning is expected to be a person's primary focus during interaction with CueFlik, we were able to present end-users with a variety of *System Feedback* information as well as *End-User Control* mechanisms.

Our first exploration (Section 4.4) examined different techniques for organizing system *Output* predictions which in turn affected end-user selection of *Input* examples to further train the system. Our results show that our *Overview*-based organization technique improved end-user ability to train accurate models compared to the best performing technique from previous work (which was shown to perform better than the standard technique used in many recommender and relevance feedback systems of providing end-users with ranked lists of examples to convey system feedback).

Our second exploration (Section 4.5) mainly experimented with the *Temporal Control Direction* design dimension by examining various techniques for providing end-user exploration of multiple potential models in CueFlik. Our evaluation results show that support for revision and exploration helped people train better CueFlik models than when revision was not available.

Many of the design techniques we presented in this chapter are not necessarily specific to the image classification problem. For example, our *Overview*-based example selection technique and support for revision could potentially be used for other domains involving example based training. Therefore, important opportunities remain to examine and extend these strategies in the context of end-user interactive machine learning in other applications.

Chapter 5
Access Control in Social Networks with ReGroup

Social networking sites present a conflict between our desire to share our personal lives online and our concerns about personal privacy (Rosenblum, 2007; Sheehan, 2002; Gross and Acquisiti, 2005). Well-known examples of privacy blunders include being fired for posting critical comments about a boss (Chen, 2010), public posting of intimate conversations (Parker, 2011), and oversharing embarrassing photos with potential employers (Korkki, 2010). Major sites have therefore begun advocating customizable friend groups as the latest tool to help control with whom we share (e.g., Ortutay, 2011 and Pogue, 2011).

The currently-advocated approach to custom group creation is to pre-categorize friends in advance of sharing decisions. For example, Google+ friends must be manually organized into "Circles" before content can be shared with them (Pogue, 2011). Katango's friend-sorting application (Kincaid, 2011) and Facebook's "Smart Lists" feature (Ortutay, 2011) attempt to aid categorization by automatically generating potential groups based on pre-defined feature combinations assumed to reflect common categories of friends (e.g., listed workplace to represent people with whom you might work with and frequency of appearance in news feeds to represent people with whom you might be close). These generated groups can then be edited for correctness.

Pre-defined groups may suffice for configuring filtered views of update streams and news feeds, but usable security recommendations argue that privacy controls for shared content should operate in context of that content (Lederer *et al,* 2004; Palen and Dourish, 2003; Smetters and Good, 2009; Whalen *et al*, 2006). This is because prior research has shown that willingness to share varies widely based on both the recipients of shared content and on the content itself (Olsen *et al*, 2005). For example, a person's definition of a "close friend" may change when sharing a personal photograph versus inviting people to a dinner party. Furthermore, Jones and O'Neill (2010) recently showed that groups created for generic

purposes only partially overlap with in-context decisions about sharing particular pieces of content. Ill-conceived groupings can therefore lead to information leakage, over-restriction, or additional work to revise pre-defined groups in-context.

In this chapter, we present ReGroup (Rapid and Explicit Grouping), a novel system that applies end-user interactive machine learning to help people create custom, on-demand groups in online social networks (Amershi *et al*, 2012). ReGroup works by observing a person's normal interaction of adding members to a group while learning a probabilistic model of group membership which it uses to suggest both additional members and group characteristics for filtering a friend list.

In the next sections, we first describe the state of the art and related work in access control and privacy in online social networks. Next, we discuss the design factors affecting end-user interaction with ReGroup. Then, we describe ReGroup and discuss the challenges we faced as a result of these identified factors in designing for effective end-user interaction with ReGroup's machine learning, including (1) how to suggest people while preserving end-user control, (2) how to indirectly train an effective classifier, (3) how to deal with unlearnable groups, (4) how to integrate human knowledge through interaction with both examples and features. Finally we present an evaluation of ReGroup with real end-users and discuss the implications of ReGroup's new approach to social access control.

5.1 State of the Art in Social Access Control

Privacy continues to be a concern in online social networks. Traditional mechanisms for specifying social network access control have been too inexpressive (e.g., allowing access to pre-defined lists of "Friends", "Friends of Friends", or "Everyone" in Facebook) or tedious and error-prone (e.g., manual selection from an alphabetical list of friends (Slee, 2007; Pachal, 2011; Siegler, 2010)). This has motivated research on alternative methods. For example, Toomim *et al* (2008) examine guard questions testing for shared knowledge and McCallum *et al* (2007) examine automatically inferring access control roles from email communication.

Recent interest has developed in automatic detection of communities within social networks (e.g., Jones and O'Neill, 2010; Kincaid, 2011; MacLean *et al*, 2011; Ortutay, 2011; Bacon and Dewan; 2011). The assumption in such work is that detected communities will overlap a person's eventual sharing needs. SocialFlows (MacLean *et al*, 2011) automatically creates hierarchical and potentially overlapping groups based on email communication. These groups can then be edited via direct manipulation and exported to Facebook and Gmail. Katango's friend sorting application (Ortutaty, 2011) and Facebook's Smart Lists

(Kincaid, 2011) feature also follow this model by providing a set of automatically generated friend groups which can then be manually edited for correctness.

Jones and O'Neill (2010) recently evaluated the feasibility of automatically generated groups by comparing manually defined groups with automatically detected network clusters. They found their automatically-detected clusters only partially overlapped with manually-defined groups by 66.9% on average, suggesting a need for considerable manual refinement. After following up with their participants, they further discovered that predefined groups only partially correspond to in-context sharing decisions. For example, in-context sharing decisions only overlapped with manually defined groups by 77.8-90.8% and only overlapped with automatically generated groups by 33.8-76.1%. In contrast, our focus is on custom, on-demand group specification via end-user interactive machine learning and is therefore better aligned with usable security recommendations (Lederer *et al*, 2004; Palen and Dourish, 2003; Smetters and Good, 2009; Whalen *et al*, 2006).

Prior work facilitating on-demand access control in social networks includes Gilbert and Karahalios's research (2009) on modeling relationship or tie strength between individuals based on network structure, communication, and shared profile data. Predicted tie strength could potentially inform privacy levels for protected content. In contrast to modeling relationships, Bernstein *et al*'s FeedMe system (2010) models user interest using term-vector profiles in order to recommend people with which to share particular pieces of content. Our work differs from these in that ReGroup iteratively learns a model based on end-user provided examples and feedback. By iteratively updating the underlying model, ReGroup can tailor its group member recommendations to the specific group a person is currently trying to create.

More similar to our approach of defining groups by example is Gmail's "Don't forget Bob!" feature, which recommends additional email recipients based on a few seed contacts and a social graph constructed from previous email (Roth *et al*, 2010). However, ReGroup provides more control over its suggestions by continuing to iteratively update its underlying model based on further examples.

Most similar to our work is Fang and LeFevre's interactive machine learning-based privacy wizard for online social networks (2010). Their wizard employs active learning to construct a privacy classifier that maps item-friend pairs to allow or deny actions. ReGroup differs in that we use end-user interactive machine learning as a tool for helping people select members of a group rather than for creating a robust classifier. As a result, ReGroup's design aims to balance end-user flexibility and control with speed as opposed to an active learning approach which prioritizes speed potentially at the expense of user experience (Baum and Lang, 1992). Furthermore, ReGroup preserves end-user desired control of group

membership (Gross and Acquisti, 2005) by requiring explicit approval of group members instead of delegating control to a classifier.

5.2 Design Factors Affecting End-User Interaction with ReGroup

Our objective with ReGroup is to help people create groups on-demand and in-context. Therefore, we envisioned ReGroup models as being disposable in that they would only be used to help a person create their current group (i.e., the *Intended Product* of the interaction is ReGroup's group member predictions). Furthermore, new models would be required for each new group to make appropriate recommendations for a person's in-context needs. In addition, a person's primary goal in using ReGroup is to create a group. Therefore, their *Interaction Focus* is on the formation of a group, not the accurate training of a model. However, ReGroup still needs to learn an accurate model in order to make appropriate recommendations. This greatly impacts our design of ReGroup's learning as will be discussed in the following sections.

A person interacting with ReGroup is assumed to have a clear idea of the group they are trying to create (i.e., the intended concept is *Well-Defined*). However, a person may not be able to easily recall everyone they wish to add. This proves to be an important benefit of ReGroup's recommendation ability (see Section 5.6 discussing the results of our evaluation of ReGroup). Also, because privacy is a primary concern in access control in social networks (e.g., sharing information with the wrong people should be avoided), we assume that it is worse to add incorrect people to a group than it is to forget to add someone. Therefore, we aim for precision over recall in ReGroup's design (i.e., precision is an important *Performance Requirement*). We also aim for efficiency in creating groups (so as to provide an improvement over the state of the art technique of manual group creation).

We assume a person will interact with ReGroup in a desktop or laptop environment where relatively large *Display* will be available to present system feedback.

In terms of constraints that must be met in ReGoup's design, ReGroup needs to operate over data available from a social network (e.g., demographic and communication information between friends). The *Nature of the Data* from a social network can often contain many missing or inaccurate fields because people often choose not to supply information, have strict privacy settings, or are inconsistent with their online activity. The features that are available about individual friends will, however, likely be familiar to a person as they reflect characteristics of those friends. The *Data Source* is assumed to be static and relatively small in *Volume* as the average person has approximately 230 friends in their online social networks (Goo, 2012).

We aim for ReGroup to be accessible by any user of a social network (i.e., no *Machine Learning Experience* is required and *Individual Differences* should not play a role). ReGroup users are also assumed to be *Domain Experts* as the data set they are interacting over are friends they are assumed to know to some degree.

5.3 ReGroup

ReGroup uses end-user interactive machine learning to help people create custom groups on demand in online social networks. In this section, we first illustrate with an example how a person can create a group with ReGroup. We then discuss the challenges inherent to interactive machine learning for group creation with respect to the design factors identified above and discuss how we address these in ReGroup.

5.3.1 Example Usage Scenario

Sally wants to advertise a confidential research talk to relevant friends at the University of Washington, so she decides to use ReGroup to create a private group for the advertisement. To start, she thinks of a friend she knows will be interested in the talk and searches for them by name (via a search box, left in Figure 5.1). She adds this friend by clicking on them in the display (right in Figure 5.1). ReGroup learns from this example and then tries to help Sally find other friends to include. It re-organizes her friend list such that friends who are more likely to be included in the group are sorted to the top of the display. Sally now sees several other friends who would be interested in the talk and adds them to her group all at once (by drag-selecting and then clicking the *Add Selected* button, right in Figure 5.1). With these additional examples, ReGroup learns more about the group being created and again reorganizes Sally's friend list to help her find additional friends.

As ReGroup learns about the group Sally is creating, it also presents relevant group characteristics she can use as filters. For example, given the currently selected friends, ReGroup believes Sally might want to include other people that live in Washington State, that tend to have several mutual friends with her, and that work at the University of Washington (top of *Filters* display on the left in Figure 5.1). Although not everybody Sally wants to include works at the University of Washington (e.g., some are students), she agrees that they all likely live in Washington. She clicks the "Current State: Washington" filter, which causes ReGroup to remove people who do not live in Washington from the ordered list of Sally's friends. This helps Sally by reducing the number of people she must consider when finding friends to include in her group.

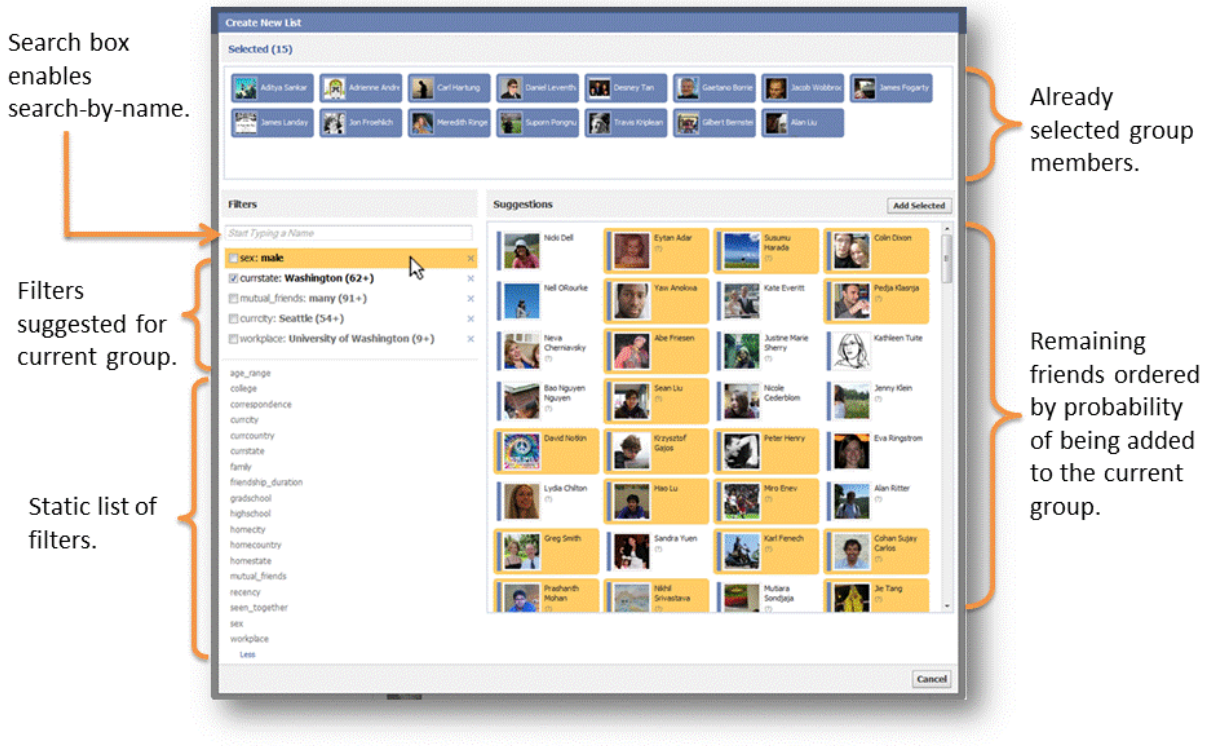

Figure 5.1: ReGroup uses interactive machine learning to help people create custom, on-demand groups in online social networks. This figure shows important aspects of ReGroup's interface. As a person selects group members (top), ReGroup suggests additional members by ordering that person's remaining friends by their probability of being added to the current group (bottom right). ReGroup also suggests relevant group characteristics for use as filters to narrow down a friend list (see five suggested filters on the top left).

Sally continues interacting with ReGroup in this way, explicitly adding group members and filtering when necessary, until she has included everyone to whom she wants to advertise.

5.3.2 Identifying Features

ReGroup's ability to suggest group members is powered by its interactive machine learning component. Machine learning works by discovering patterns in examples. A system is therefore strongly influenced by the quality of information contained in the representations of those examples (i.e., the features).

We based our features on related work on social networks (e.g., Gilbert and Karahalios, 2009). We also conducted an online survey of Facebook and Google+ users to identify common groups we could support. We distributed the survey to our own Facebook and Google+ contacts, obtaining 69 responses (21 Facebook and 48 Google+) describing 244 custom groups (32 Facebook and 212 Google+). Facebook and Google+ advocate creating custom groups in advance, so the groups identified in our survey may not resemble groups people would create with an on-demand system (i.e., the composition of permanent

Table 5.1: This table lists the 18 features used by ReGroup's interactive machine learning (left) along with examples of groups each feature may help support (right).

Feature	Description/Examples
Gender, Age Range	"Sorority friends", "People my age"
Family Member	"People I'm related to"
Home City, State and Country	"Friends from my hometown"
Current City, State, Country	"Anyone I know who lives nearby"
High School, College, Graduate School	"Anyone from my department at school"
Workplace	"CoWorkers"
Amount of Correspondence	Number of Inbox, Outbox and Wall messages sent and received. "People I socialize with regularly"
Recency of Correspondence	Time since latest correspondence. "Friends I currently interact with"
Friendship Duration	Time since first correspondence [X]. "Friends I've known over 10 years"
Number of Mutual Friends	"My group of closest friends"
Amount Seen Together	Number of photos a person and friend are both tagged in and number of events in which both were invited. "People I see on a regular bases"

groups may differ from groups created in context). However, these gave a starting point from which to distill potential features.

Table 5.1 presents the 18 features currently used by ReGroup, together with an example group the feature might help support. In this research, we endeavored to support common groups as part of demonstrating the potential for interactive machine learning in social access control. Our features are therefore meant to enable our research with ReGroup and are not intended to be exhaustive.

5.3.3 Suggesting People while Preserving End-User Control

ReGroup's group member suggestions are realized by a Naïve Bayes classifier (Domingos and Pazzani, 1997). Each friend is represented by a set of n feature-value pairs (Table 5.1). The probability of each friend being a member of the group g is then computed via the following Bayesian formulation:

$$P(g|x_1, x_2, \ldots, x_n) = \frac{P(x_1, x_2, \ldots, x_n|g)}{\sum_{g'=g,\bar{g}} P(x_1, x_2, \ldots, x_n|g')P(g')}$$

where $P(x_1, x_2, \ldots, x_n|g)$ is the likelihood of a friend with feature values $x_1, x_2, \ldots, x_n$ being a member of g, $P(g)$ is the prior probability of any friend belonging to g, and the denominator is the probability of seeing the set of feature values in the data and serves as a normalizing constant.

The Naïve Bayes assumption considers each feature to be conditionally independent given the class, reducing the likelihood computation to:

$$P(x_1, x_2, \ldots, x_n|g) = \prod_{i=1}^{n} P(x_i|g)$$

where the probability of a group member having a particular feature value, $P(x_i|g)$, can be estimated by a frequency count of the number of current group members having that feature value over the total number of current group members. We also use Laplace smoothing here to improve performance in the presence of limited training data.

Although the independence assumption is often violated in real-world data, Naïve Bayes has been shown to work well in many practical applications (Domingos and Pazzani, 1997). Naïve Bayes also gracefully handles missing data and allows for a straightforward interpretation of features, both important design factors impacting ReGroup (i.e., *Nature of the Data* as discussed in Section 5.2).

ReGroup's classifier is re-trained every time a person adds friends to a group. ReGroup then reorders a person's remaining friends according to who is most likely to also belong to the group as computed by the updated classifier. To preserve end-user control during interaction with ReGroup's classifier, we also made the design decision that people must explicitly approve every group member before they are added to a group. This decision also better supports the precision *Performance Requirement* than having ReGroup automatically add suggested friends.

5.3.4 Indirectly Training an Effective Classifier

Effective training of machine learning systems requires both positive and negative examples. Therefore, a system focused on training an effective classifier will be designed to solicit explicit positive and negative examples (e.g., as in CueFlik discussed in Chapter 4). However, as discussed in Section 5.2, a person's primary goal with ReGroup is to create a group (i.e., their *Interaction Focus* is on group creation). Therefore, the classifier is a disposable side effect, and a person is not concerned about its generalization

(i.e., the *Intended Product* of interaction is ReGroup's group member predictions, not the model itself). This was confirmed in our study presented later in this chapter, wherein participants primarily scanned the interface for positive examples (we observed 2077 positive examples compared to only 89 negative examples).

To mitigate the effects of people primarily providing positive examples, we designed ReGroup to obtain implicit negative examples during interaction process (see *Implicit End-User Control* in Section 3.2.2). When a person selects a group member from an ordered list of friends, ReGroup increases the probability that the skipped friends (i.e., friends preceding the selection in the ordered list) are *not* intended for the group. This is achieved by assigning the preceding friends implicit negative labels for use in ReGroup's group membership computations. However, this heuristic is not always correct (e.g., a person's gaze may be immediately drawn to a friend further down in the list, without having viewed or decided upon the preceding friends). Therefore, an implicit negative example contributes only a partial frequency count, α, in ReGroup's computation of $P(x|g)$. ReGroup also still computes a group membership probability for each implicitly labeled friend and ranks them. We found that setting α=0.2*n, where n is the number of times a friend is skipped, worked well in practice.

5.3.5 Minimizing Frustration Caused by Unlearnable Groups

While experimenting with early versions of ReGroup, we observed that skipped friends would sometimes continue bubbling back up in the ordered list during group creation. Further examination revealed that this occurred when a person had all the characteristics of the group, as modeled by the classifier, but was skipped for some reason not captured by the system (e.g., a person planning a surprise party would obviously not invite the guest of honor). Implicit negative examples are not powerful enough to prevent repeated suggestion of such friends because of their high similarity to the many positive examples.

This problem occurs in all machine learning systems when the hypothesis language is insufficiently expressive to model the true concept (i.e., the features in the data are not powerful enough to represent all concepts). For interactive machine learning in ReGroup, however, it can cause serious frustration as a person can quickly grow tired of repeatedly skipping the same friend. We addressed this with an explicit penalty term, v, in our group membership estimation as follows:

$$P(g|x) = P(g|x) * v^n$$

where n is the number of times a friend was skipped. Our preliminary experiments showed that setting v=0.9 achieved the desired effect (i.e., reduced the likelihood of a skipped friend continuing to be highly ranked when ReGroup could not learn the true concept).

5.3.6 Decision-Theoretic Filter Suggestions

Most end-user interactive machine learning systems focus only on interaction with examples. However, because we assume that people are familiar with characteristics of their friends (i.e., they are *Domain Experts*) and have knowledge about the shared properties of the groups they are trying to create (i.e., they have defined concepts in mind, *Concept Flexibility*), we hypothesized that enabling interaction with features of the machine learning might accelerate the group creation process.

We chose to realize end-user interaction with features in ReGroup using faceted search (Hearst, 2006), a popular method for helping people find objects in collections (e.g., www.amazon.com, www.yelp.com). With faceted search, people find objects by filtering a collection based on feature values. ReGroup provides two ways for people to filter friends by feature values:

- Via a suggested list of five top feature value filters (top of *Filters* display, left in Figure 5.1).
- Via a static, hierarchical list of all feature value filters (bottom of *Filters* display, left in Figure 5.1).

Previous methods for ordering filters have used metrics like hit count (e.g., Hearst, 2006), query keywords (e.g,. www.yelp.com), query logs and click through data (e.g., van Zwol *et al*, 2010), navigation cost (e.g., Li *et al*, 2010), or user models defined by explicit ratings of documents (e.g., Koren. 2008). However, none of these are appropriate for our problem domain. We therefore formulate the problem decision-theoretically.

Intuitively, we want ReGroup to suggest filters that will reduce effort required during group creation. A suggested filter must therefore represent the intended group well (i.e., $P(x_i|\, g)$ must be high). However, this alone does not guarantee that a filter will prune unwanted friends. We therefore combine this with the expected utility of a filter as follows:

$$U(x_i) = P(x_i|\, g) * InformationGain(X_i)$$

Here, we use information gain (Mitchell, 1997) to approximate the potential time savings of activating a filter. That is, features with high information gain tend to have a larger number of distinct feature values distributed across the data. Therefore, the higher the information gain of a feature, the more likely the corresponding filter will prune unwanted friends (i.e., by eliminating people with feature values different from the value represented by the filter). We use this formulation in choosing the top five filters to suggest, as well as in reordering feature values within the static list.

5.3.7 Gracefully Handling with Missing Data

As discussed in the section on design factors affecting ReGroup (Section 5.2), missing data is rampant in the domain of online social networks. This can lead to unpredictable behavior in machine learning systems. The Naïve Bayes classifier gracefully handles missing data by ignoring features with missing values when computing group membership probabilities. However, missing data can still adversely impact the usefulness of ReGroup's filters. For example, missing data could result in incorrectly filtered friends after applying a filter (e.g., when filtering on "Seattle", a friend who lives in Seattle might be filtered because they have not supplied location information to their social networking site). Missing data could similarly result in incorrectly preserved friends after filtering (e.g., when filtering on "Seattle", a friend who lives in Austin might be retained because they have not supplied any location information).

ReGroup attempts to estimate missing values from the data by using a classifier to predict corresponding feature values conditioned on all other available features. Then, when applying a filter, ReGroup only eliminates friends who are *guaranteed* to be ineligible (i.e., have provided some value other than the filter value). For friends with missing data, ReGroup indicates its uncertainty in its data estimates by displaying a question mark under a person's name in the interface (see Figure 5.1). If a person hovers over a question mark, ReGroup displays a tooltip showing its guess for the corresponding filter value for that friend. ReGroup thus avoids incorrectly eliminating friends while reducing irritation by displaying guesses for incorrectly retained friends.

5.3.8 Implementation Details

ReGroup is implemented using Facebook's Application Platform (http://developers.facebook.com/) and a Firefox Greasemonkey script (http://www.greasespot.net/). ReGroup uses Facebook's Application Platform to access relevant information about a person and their friends (pertaining to the 18 features used by ReGroup's interactive machine learning listed in Table 5.1), while the Greasmonkey script allows ReGroup to run within a person's own Facebook account.

ReGroup is not publicly available (i.e., only people participating in our evaluation could access the Facebook Application and use the Greasemonkey script installed on the study computer in our lab). ReGroup accessed a participant's Facebook information only after they provided explicit and informed consent and only for enabling its interface, staying within Facebook's Terms of Service.

5.4 Evaluation

We conducted an evaluation to explore tradeoffs between end-user interactive machine learning for on-demand custom group creation and Facebook's current approach of allowing manual selection from an

alphabetical list or searching by name (Slee, 2007). We also wanted to compare a design including our feature-based interaction with a more typical design using only example-based interaction. That is, we particularly wanted to evaluate *End-User Control* techniques involving both *Input* examples and *Features* compared to control involving only *Input* examples.

Interface Conditions

We evaluated the following interfaces:

- *Alphabet*. A person can scroll through an alphabetically ordered list of friends, selecting people as they scroll. They can also search by name. This is equivalent to Facebook's current on-demand, custom group creation.

- *Example-Only.* Each time a person adds a friend to the group, the list of friends is reordered based on ReGroup's current estimation of group membership probability. People can also still search for friends by name.

- *Example-Attribute*. The full ReGroup design, enhancing the *Example-Only* with our decision-theoretic techniques for feature-based interaction.

Study Design and Procedure

We ran a within-subjects study, counterbalancing order of interface conditions using a Latin square design. At the beginning of the study, we told participants they would be testing new tools for helping people create custom groups in Facebook. We also explained the tools would work in their own Facebook accounts and that they would be testing the tools by creating groups of their own friends. At this point, participants were informed about what data would be accessed and how it would be used for our evaluation and to enable the interfaces during the study. We continued only after they provided written consent and granted our Facebook Application permissions to access their data.

Next, prior to seeing any interface, participants were asked to think of six groups they could create during the study. We chose this approach to ensure groups were meaningful, as assigned groups may not correspond to distinctions a person would make among their friends (i.e., artificial groups would likely impact *Concept Flexibility*). When thinking of groups, participants were instructed to imagine they were about to post a new comment or share a new photo and only wanted to share it with a select group of friends. We also provided them with a list of ten example groups based on frequent responses to our online survey (e.g., "home town friends", "close friends"). Participants were told they could select from this list, create a variation, or create entirely different groups.

Resulting groups ranged from typical (e.g., “Family”, “CoWorkers”) to more unique and nuanced (e.g., “Older Faculty Members”, “People I Care About”).We also asked participants to estimate group size (“small”, “medium” or “large”). Estimated size was used by the experimenter to balance groups with interface conditions. The experimenter sorted groups by estimated size and assigned them to conditions in order (thus evenly distributing groups across conditions by size).

The experimenter then demonstrated ReGroup (using the full *Example-Attribute* interface) with the experimenter’s own Facebook account. Participants were told they would use three variations of the ReGroup interface. Before each condition, the participant practiced with the corresponding interface by creating a simple gender-based group. After the practice, participants created two of their groups, one at a time (they were not told which groups they would create until they were about to create them). To avoid exhausting participants, we limited each group to a maximum of 4 minutes (we did not inform participants of this limit, but simply asked them to stop if they reached it).

All interface actions were time-stamped and logged. All participant information and logged data was stored anonymously, using unique identifiers, on a secure server accessible only by the researchers. After each group, participants completed a short questionnaire containing 5-point Likert scales (1=strongly disagree, 5=strongly agree) about the group they just created in the interface they just used (e.g., “I was happy with the group I just created”). At the end of the study, participants filled out a final questionnaire asking them to compare the different versions they tested (e.g., “Rank the versions to indicate which was your favorite”) and to comment on their overall experience. The study lasted 1 hour and participants were given a $20 Amazon gift certificate for their participation.

Participants and Data

Twelve people (four female, ranging in age from 18-34) were recruited for the study via a call for participation sent to relevant university mailing lists. As a result, all of our participants were technically savvy, but ranged from undergraduates to staff members from a variety of disciplines (e.g., Computer Science, Psychology, Design). We required participants have an active Facebook account they had used for at least one year with at least 100 friends. This was to help ensure participants would be active enough on Facebook for ReGroup to be potentially useful.

As expected, participants varied in their composition and number of friends (mean=385.4, min=136, max=781) and their activity on Facebook (ranging from every day to a few times a month). Our participants also showed an average of 36.3% missing data with respect to ReGroup’s implemented features.

5.5 Results

We performed all of our log and Likert scale data analyses using a nonparametric repeated measures analysis of variance, after aligning the data according to the *aligned rank transform* (Wobbrock *et al*, 2011) to preserve interaction effects due to having participants create two groups per condition. We also performed post-hoc pairwise comparisons when a significant effect was observed. To analyze our final ranking questions, we used a randomization test of goodness-of-fit (Manly, 1997) which is more robust against smaller sample sizes than a standard Chi-Square test. For each test, we ran 10,000 Monte Carlo simulations. Table 5.2 and Table 5.3 show the per-condition means and standard deviations for all metrics used in our log and Likert scale data analyses, respectively. Table 5.3 also shows the number of

Table 5.2: Mean and standard deviations of all metrics used in our log data analysis. Metrics with *'s indicate a significant effect was observed.

	Alphabet	***Example-Only***	***Example-Attribute***
*Final Time**	163.0/63.4s	196.9/56.8s	216.0/35.8s
Final Group Size	25.3/24.8	34.0/40.6	27.2/22.1
Mean Select Time	8.6/3.9s	13.9/16.3s	15.9/16.5s
*SD Select Time**	9.4/6.5s	13.5/8.5s	19.7/15.1s
*Mean Position**	171.8/87.2	58.1/78.9	38.5/48.8
*SD Position**	101.7/56.1	41.1/33.3	40.5/41.1
Single Selections	18.0/27.0	8.5/6.2	7.9/5.1
*Multi-Selections**	0.4/0.8	2.8/2.9	2.8/3.0
Search-Selections	4.3/4.7	2.1/2.7	2.0/2.1

Table 5.3: Likert mean and standard deviations (1=strongly disagree, 5=strongly agree) and ranking (number of participants) responses. Metrics with *'s indicate a significant effect was observed.

	Alphabet	***Example-Only***	***Example-Attribute***
Happiness	3.9/0.9	3.9/1.1	4.0/0.7
*Easiness**	2.6/1.0	3.2/1.2	3.3/0.8
*Quickness**	2.3/1.0	3.0/1.2	3.4/1.0
*Favorite**	1	0	11
*Best Helped**	3	1	8

participants choosing each condition for each of our ranking questions. We discuss all of our quantitative analyses in the context of our qualitative observations and feedback from participants.

We analyze our study data in terms of the overall time taken and final group sizes, the speed and effort of selecting group members, and interface element usage. Note that we cannot evaluate group accuracy because no adequate ground truth is available or obtainable. Asking participants to re-create Facebook or Google+ groups could bias their notion of those groups. Alternatively, asking participants to verify group completeness would require exhaustive labeling (recognized as error-prone and analogous to list scrolling in the *Alphabet* condition).

Final Times and Group Sizes

Overall, participants created 72 groups with a total of 2077 friends. Examining the *Final Time* taken to create groups, our analysis shows a significant effect of interface condition ($F_{2,55}$=6.95, p≈.002). Post-hoc pairwise analyses reveal that participants using the *Alphabet* interface took significantly less time to create groups than when using both the *Example-Only* ($F_{1,55}$=5.93, p≈.018) and *Example-Attribute* ($F_{1,55}$=13.4, p≈.0006) interfaces. There was no difference in *Final Time* between *Example-Only* and *Example-Attribute* conditions. One explanation for participants taking less time in the *Alphabet* condition is that both of the reordering conditions (*Example-Only* and *Example-Attribute*) required additional time to update the display when reordering or filtering. Another contributing factor could be that participants in the *Alphabet* condition often attempted to recall friends by name to avoid scrolling through their entire list of friends (e.g., *"I was surprised how useless alphabetical ordering is, but keyword search was very useful"*). This may have resulted in people forgetting to include some friends and stopping early. Interestingly, participants who resorted to scrolling through the full list often felt like they missed people (e.g., *"umm, I guess that's it"*, *"there's probably more, but oh well"*). One participant also explicitly commented *"It's too easy to forget about people when it's ordered alphabetically."*

Difficulty recalling friends in the *Alphabet* interface could have resulted in the shorter *Final Times* in that condition. However, one would then also expect to see larger *Final Group Sizes* in the reordering conditions because of their ability to surface relevant people, favoring recognition over recall (e.g., *"Reordering makes it <u>much</u> faster than alphabetical because it reminds you of people without having to do an exhaustive search"* and *"Reordering helps me quickly pick friends in the first rows. Filters keep me from frequent scrolling"*). However, our analysis showed no significant difference in *Final Group Size* across conditions. Further analysis showed the presence of a ceiling effect in all conditions, suggesting that participants were often cut short of the time they needed to complete their groups, which could account for the lack of difference in *Final Group Size*. This effect was also more pronounced in the

reordering conditions (16.7%, 33.3% and 45.8% of interactions were cut short in the *Alphabet*, *Example-Only*, and *Example-Attribute* conditions, respectively).

Speed and Effort in Selecting Group Members

To compare the time between group member selections, we had to account for the fact that participants could add friends individually or in bulk (i.e., a multi-select action). In the case of a multi-select, we assigned the time between the action and the previous group member selection to the first friend in the multi-selection and assigned a time of zero to the rest of the friends. We did not see an effect of interface condition on *Mean Select Time*. We did however see a difference in *SD Select Time* ($F_{2,55}$=7.78, p≈.001), with post-hoc pairwise comparisons showing significant or marginal differences in all cases*: Alphabet* was less than *Example-Only* ($F_{1,55}$=2.83, p≈.09) and *Example-Attribute* ($F_{1,55}$=15.5, p≈.0002), and *Example-Only* was less than *Example-Attribute* ($F_{1,55}$=5.07, p≈.03).

Although we did not see an effect of condition on *Mean Select Time*, the average *Mean Select Time* was smallest in *Alphabet* (see Table 5.2). This is partially an artifact of the time needed to update the display in the reordering conditions. In *Example-Attribute*, a display update can also occur as a result of a filtering action. As this confounds our analysis of *Select Time*, we decided to measure the position in the display of each group member immediately before selection as a proxy for effort level. That is, if position is low, a friend was closer to the top of the display and therefore required less effort to locate. We saw a significant effect of condition on *Mean Position* ($F_{2,55}$=39.1, p≈.0001), with post-hoc pairwise comparisons showing that the *Mean Position* in *Alphabet* was significantly greater than in both *Example-Only* ($F_{1,55}$=47.8, p≈.0001) and *Example-Attribute* ($F_{1,55}$=67.7, p≈.0001). We also saw a significant difference in consistency of position as measured by *SD Position* ($F_{2,55}$=19.6, p≈.0001), again with *SD Position* being significantly greater in *Alphabet* compared to both *Example-Only* ($F_{1,55}$=2.81, p≈.0001) and *Example-Attribute* ($F_{1,55}$=31.8, p≈.0001). Lower positions indicate that people had to scroll less to search for friends in the reordering conditions because these conditions sorted potential group members closer to the top of the display for easy recognition and access. As expected, the *Mean Position* of selected friends in the *Alphabet* condition was roughly half, 44.6%, of the average number of Facebook friends of our participants and their *SD Position* was highly varied because group members were often evenly distributed throughout the entire alphabetical list.

Our questionnaire results provide additional evidence of reduced effort in the reordering conditions. Analyses of our Likert scale questions show an effect of condition on perceived levels of *Easiness* ($F_{2,55}$=4.33, p≈.018), with the both the *Example-Only* and *Example-Attribute* interfaces being perceived as easier to use than the *Alphabet* interface ($F_{1,55}$=5.74, p≈.02 and $F_{1,55}$=7.18, p≈.01, respectively). Similarly,

we found a significant effect of condition on perceived *Quickness* ($F_{2,55}$=6.63, p≈.003) in creating groups, again with *Example-Only* and *Example-Attribute* interfaces being perceived as quicker than *Alphabet* ($F_{1,55}$=5.80, p≈.02 and $F_{1,55}$=12.74, p≈.0008, respectively). We saw no difference in terms of perceived *Easiness* or *Quickness* between *Example-Only* and *Example-Attribute.* Perceived *Easiness* and *Quickness* in the reordering conditions is likely due to these interfaces automatically surfacing relevant friends (e.g., "*Sometimes it really looked as if the system was reading my mind!"*).

Behavioral Differences and Feature Usage

In all interfaces, participants could select friends to include one at a time (*Single Selections*), several at a time (*Multi-Selections*), or by searching for them by name (*Search-Selections*). Our participants used all of these features in each condition, however they searched for friends by name and added friends one by one less often in the reordering conditions compared to the *Alphabet* condition. In addition, we found a significant effect of condition on the number of *Multi-Selections* ($F_{2,55}$=10.9, p≈.0001) with the *Example-Only* and *Example-Attribute* conditions both showing increased *Multi-Selections* compared to the *Alphabet* condition ($F_{1,55}$=17.4, p≈.0001 and $F_{1,55}$=15.4, p≈.0002, respectively). Increased *Multi-Selections* and fewer *Single* and *Search-Selections* in the reordering conditions is likely due to these interfaces making friends easier to add as a group by sorting relevant friends to the top of the display and filtering out irrelevant friends (in *Example-Attribute*).

Our logged data also showed that participants used the suggested and static filters when they were available (in *Example-Attribute*). Participants selected *Suggested Filters* 1.9 times on average (*SD*=1.3), selected *Static Filters* 0.9 times on average (*SD*=1.3), and *Unselected Filters* that they had previously selected 2.3 times on average (*SD*=2.8). Our observations showed that participants sometimes used the filters in clever and unexpected ways, such as selecting them temporarily to find a certain set of friends and then unselecting them to find others. One participant commented that the "*Filters helped me guide the tool"*. Other comments indicate that the suggested filters served as an explanation of how the system was working (e.g., *"The filters helped me understand what patterns the tool was discovering, and thus helped understand the tool's behavior"*). This indicates that along with helping people find group members, filter suggestions can also serve as a means for providing *System Feedback* on some of the *Logic* behind ReGroup's interactive machine learning.

5.6 Discussion and Future Work

Although a significant majority of our participants rated the full ReGroup interface (i.e., *Example-Attribute*) as their overall *Favorite* (χ^2=18.5, p≈.0001) and felt that it *Best Helped* them create groups (χ^2=6.5, p≈.04), participants did become frustrated when ReGroup was unable to model the group well, as

this essentially resulted in having to search through an unordered list for group members (e.g., *"Ordering is useful for some lists, annoying for others"* and "*though I liked the filters, I think they didn't work in some cases"*). As with all machine learning based systems, the ability to model a concept is highly dependent upon the quality of the underlying data (e.g., the expressiveness of the features and the availability of data). Additional features should improve ReGroup's performance in this regard, but missing data is still inherent to social networks. Future work might therefore aim to detect such cases and degrade gracefully (e.g., by suggesting an end-user resort to traditional methods for searching for group members).

Interestingly, some participants could tell why ReGroup was not working well in some cases (e.g., *"Groups that all have something in common that is tracked by Facebook (i.e., college) were easier to create than a group of people who are seemingly unrelated based on Facebook information."*). This indicates that people can understand certain aspects of interactive machine learning-based systems (in this case the impact of the data on a system's capabilities). This therefore also suggests further exploration of the potential to exploit *Human Learning* to enable more effective use of these systems (e.g., by enabling end-users to determine when and interactive machine learning system can help versus when it is incapable of doing so).

While ReGroup's group member recommendations helped reduce the amount of effort required to find friends to add to a group, we observed that some people became frustrated with ReGroup's shuffling of friends after every interaction. One participant commented that "*reordering is often frustrating - occasionally the person I want to add disappears from the top of the suggestions list just because I clicked on someone first.*" This suggests re-evaluation of our approach to the *Timing of System Updates* design dimension which we set to update after every user action. That is, because a person's primary *Interaction Focus* with ReGroup is to create a group, the disruption caused by reordering too frequently (regardless of whether it accurately reflects an update to the system) could increase end-user frustration and therefore should be mitigated in the future.

Overall, our evaluation showed that different interfaces worked well for different kinds of groups (e.g., *"[My favorite version] depended on the group I was creating"*). We observed, and participants reported, that the *Alphabet* condition appeared to work well for small groups whose members could be easily recalled by name (e.g., family members with the same name, co-authors of a paper, and members of a cross-country team). In contrast, the reordering conditions worked better for larger and more varied groups (e.g., childhood friends, local friends or friends from particular regions of the world, college friends, former and current colleagues, people in a particular field of work, and friends with a professional

relationship). One participant noted that for *"most of the small groups that I wanted to create, I already knew the list in my head [so] alphabetical was easiest. However, for larger groups, reordering was most efficient."* A likely result of this is that we found no significant difference in terms of overall participant *Happiness* with their final groups created in each condition. Therefore, we advocate integrating ReGroup's novel interactive machine learning techniques together with traditional methods used in online social networks to support a wider range of potential groups.

Also, it should be noted that although a person's primary focus in interacting with ReGroup is on creating groups rather than on training a robust classifier, a classifier as a by-product of group creation could enable automatic group maintenance as relationships change over time. Therefore, future work might evaluate the accuracy of ReGroup's classifiers for group maintenance and design additional interaction techniques to support this task.

5.7 Summary

This chapter has presented ReGroup, a novel system we developed that employs end-user interactive machine learning to help people create custom groups on-demand in online social networks. Our work with ReGroup addressed several design challenges associated with interactive machine learning in the domain of social access control (e.g., dealing with missing data inherent to social networks). In addition, we specifically experimented with support for *End-User Control* over both *Input* examples (i.e., group members) and *Features* (i.e., filters), finding that people were able to and liked interacting with both *Control Types* because of their familiarity with the underlying data.

Overall, our evaluation of ReGroup showed that different group creation techniques helped in creating different kinds of groups. For example, the traditional search-by-name approach is efficient for small, well-defined groups where members are easy to recall by name. ReGroup's support for recognition over recall, embodied by both its group member and filter suggestions, helps for larger and more varied groups. Therefore, we advocate integrating ReGroup's novel techniques together with traditional methods used in online social networks to support a wider range of potential groups.

Chapter 6
Computer Network Alarm Triage With CueT

Triaging alarms is the first line of defense for modern computer networks. Triage is the process by which the multitude of low-level alarms (e.g., high link utilization, fan failure) generated by individual devices are grouped and prioritized. These groups are then inspected by network engineers for problems with the network. Grouping related alarms, which likely stem from the same problem, helps engineers by reducing the number of issues they need to investigate and giving them a broader view of the problem than what an individual alarm provides. It is critical that triage is fast and accurate so engineers are not misled and problems can be identified and resolved quickly.

Many automated systems have been developed for alarm triage (see Gardner and Harle, 1996 or Steinder and Sethi, 2005 for reviews). However, despite years of effort, these systems are never fully accurate due to the complexity of the problem. Large networks have thousands of diverse devices, each generating a different set of alarms. Further, because each network is different and the set of devices within a network changes with time, it is very challenging to develop systems that work across networks. Therefore, to cope with the inherent inaccuracy of automated systems, networks invariably employ human operators (so called “Tier 1” operators) for alarm triage. These operators are required to sift through the thousands of alarms per day that can be missed by automation.

In this chapter, we explore a fundamentally different approach to alarm triage – end-user interactive machine learning. That is, because human operators need to be involved in the triage process anyway, we ask if we can learn from their actions and in turn use that learning to assist them. By learning continuously and in situ from operator actions, the assistance provided could better fit the network and its unique practices as well as evolve with the network.

This chapter is organized as follows. First we describe the state of the art and related work in alarm triage. Here, we also describe our own observations of the triage process in the network operations center of a large organization that we were working with. Then, we describe the design factors affecting end-user interaction with machine learning for interactive alarm triage. Next, we present CueT, a novel system we developed that uses interactive machine learning to help network operators during the triage process (Amershi *et al*, 2011). Unlike most prior interactive machine learning systems, CueT operates in a dynamic environment where the data and classes of interest are unknown *a priori* and evolve constantly. Therefore, in describing CueT, we also discuss the unique challenges of designing end-user interactive machine learning under these constraints. These challenges include: (1) making recommendations for a stream of data and constantly evolving groups, and (2) visualizing recommendation quality to reduce operator errors. We then present an evaluation of CueT with real network operators and discuss the results and implications for our findings.

6.1 State of the Art in Alarm Triage

Most research in network monitoring and alarm correlation has focused either on visualizations or automated solutions in isolation. Researchers have recently proposed visualization systems for network monitoring and diagnosis (e.g., Fisher *et al*, 2008, Lakkaraju *et al*, 2004). For example, Fisher *et al*'s Visual-I system (2008) uses visual grouping and scatterplots to highlight correlations between multiple devices and problems. While these systems leverage human visual and cognitive abilities to process data and look for patterns, our approach combines visualization and interactive machine learning in order to further reduce the cognitive effort of manually identifying patterns in large data sets. Helping automate pattern discovery can increase operator efficiency and accuracy, necessary for these time critical problems.

Automated alarm correlation (also related to root cause analysis and fault localization) has long been an active area of research because of the complexity of the problem and the potential impact on the industry (Gardner and Harle, 1996; Steinder and Sethi, 2005). Most solutions to this problem have taken the form of expert defined rules or models (e.g., Appleby and Friedman, 2001; Brugnosi *et al*, 1993; Gardner and Harle, 1996; Jakobson and Weissman, 1993; Liu *et al*, 1999). Such approaches require manual configuration of rules which can be difficult to obtain, are not robust to new situations, and require frequent maintenance (Gardner and Harle, 1996; Steinder and Sethi, 2004). In contrast, our interactive machine learning approach can handle general alarm correlations that require alarm grouping based on similarity.

Some researchers have explored automatically learning rules or models from data that can then be used for automated alarm correlation and filtering (e.g., Klementtinen *et al*, 1999; Steinder and Sethi, 2004). However, most of these approaches require extensive training periods and must be retrained whenever the network topology changes (Steinder and Sethi, 2004). In contrast, our approach is based on a dynamically changing model that is constantly learning from human guidance. We argue that human input is critical during the alarm correlation process because of the inherently inconsistent and ambiguous nature of the problem (Gardner and Harle, 1996), which is a contributing factor to why alarm correlation is still an open area of research.

While today's commercial systems employ some of these previous techniques for alarm triage and filtering (e.g., EMC Ionix, HP OpenView, Yemini *et al*, 1996), most companies still require teams of Tier 1 operators to manually triage the thousands of remaining alarms unhandled by these existing systems. We visited the network operations center of one such large organization on multiple occasions to understand current triage practices with one of these commercial systems. We first talked informally with the center's managers to get a broad understanding of the process. We then closely observed two Tier 1 operators (1 female) for an hour each as they were triaging alarms using the current tool employed at this network operations center, and asked them questions during and after the observations.

The current tool used at the network operations center that we studied is a combination of two pieces: (1) an automated rule-based engine that selects alarms to be shown to Tier 1 operators and labels those alarms (e.g., alarm severity level) and (2) a table-based UI for operators to triage alarms using labels and other features (see Figure 6.1).

Our investigation revealed three main failings of the current process. First, the rule-based engine is hard to configure and keep up-to-date as the network evolves. As a result, thousands of alarms must still be manually triaged by operators every day. While triaging, the operators largely ignored the tool's output, except for some basic information like alarm severity that is used for prioritizing which alarm to focus on next. Much of this information is not computed by the tool but is directly configured by network managers.

Second, the triage process is highly manual in part because network managers often have specific instructions on how operators should temporarily handle certain types of alarms. These instructions are based on how managers expect the network to behave as they are making changes behind the scenes. They are typically conveyed in a formal document known as a *whiteboard* which is updated daily and which operators are instructed to review before beginning a new workday. Such special case instructions

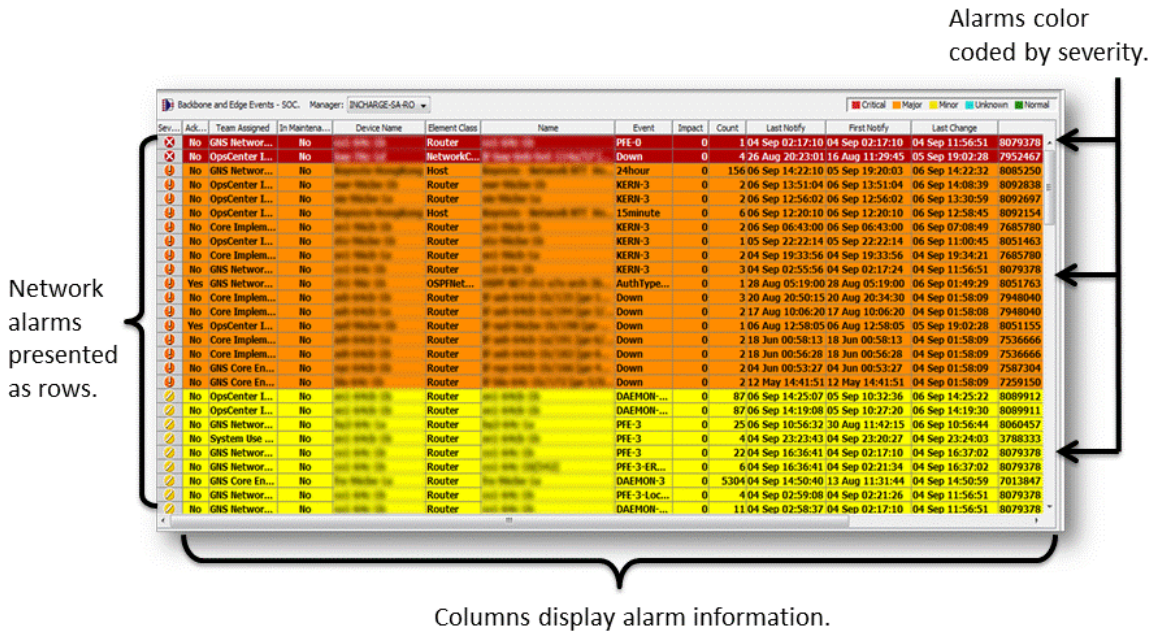

Figure 6.1: Current state-of-the-art tool used for displaying computer network alarms in a network operations center we observed. Each network alarm is presented as a row in the table and is color coded by severity. A separate tool with a similar interface is used for displaying tickets.

make it hard to develop tools for assisting with alarm triage. This difficulty applies not only to the traditional methods, but also to our approach as short-term special cases create noise in our learning model.

Third, the table-based visualization for the current triage process is rudimentary and inefficient. The alarms are presented as rows in a table that can be ordered by alarm attributes (e.g., severity). Another table in a different window shows the recently created alarm groups. The operators need to switch between these two windows to handle an alarm.

In addition, we learned that some alarm attributes are more important than others for the triage process. Understanding the importance of alarm attributes helps us not only choose the right features for the machine learning algorithm but also design appropriate visual representations (see Section 6.3 below).

Although we studied only one network in detail, our conversations with network managers (who have a broader view of the industry and have also worked with other networks) confirm that the processes at this network are similar to those of many others in operation today.

6.2 Design Factors Affecting End-User Interaction with CueT

The alarm triage process involves grouping and prioritizing a continuous stream of alarms. Moreover, new problems are constantly arising manifesting in new groups that must be created and are unknown *a priori*. Therefore, CueT's underlying model must continually evolve to handle these new problems (i.e., the *Evolutionary Needs* of the model are significant).

In addition, it is assumed that a network operator's primary goal in using CueT is to accurately triage alarms. That is, the operator's *Interaction Focus* is on making triage decisions on each incoming alarm (as opposed to a focus on training an accurate model). In addition, triage accuracy and speed are vital in the alarm triage process (i.e., CueT's *Performance Requirements* include both accuracy and speed). Therefore, CueT must be designed to help operators make accurate triage decisions quickly.

As with our previous systems, we expect operators to interact with CueT via a desktop or laptop environment with a large *Display Size* available for communication.

CueT must operate over streaming network data (i.e., the *Data Source* is streaming). In addition, the *Nature* of network data yields a variety of heterogeneous features pertaining to information available from network alarms. These features include the device name, event type, and alarm severity (as discussed further in our presentation of CueT). The *Volume* of alarms that the system and operator must handle is also relatively large (on the order of 1000s of alarms per day).

The operators themselves are not expected to have any *Machine Learning Experience* but do have *Domain Expertise* about computer networks in general, the overall configuration of their particular network, and the alarm triage process. These operators are also triaging as part of their employment and are being paid for their time (i.e., they may not be intrinsically *Motivated* to accurately or efficiently triage).

It should be noted that many operators might be triaging alarms at any given time. However, we did not consider the *Collaboration* factor in designing CueT, leaving design innovation for groups of operators as future work.

6.3 CueT

CueT consists of two complementary components: (1) a stream-based interactive machine learning engine for making triage recommendations and (2) an interface to assist operators in inspecting and interpreting those recommendations and feeding operator actions back into the learning engine. We will refer to the alarm currently being triaged as the "incoming alarm," a group of one or more related alarms that have

already been triaged as a "ticket," and the current set of tickets that are being used for recommendations (and have not been discarded yet) as the "working set of tickets."

6.3.1 Stream-Based Interactive Machine Learning

One key element of CueT's interaction scenario is that it resides in a dynamic environment, where both the alarms and tickets being generated are constantly changing. Thus, our scenario is fairly different than the classical machine learning setting, where the classes are fixed and known *a priori* and the classifier operates in an assumed static environment. We tackle these challenges due to the dynamic environment and ever evolving set of classes (i.e., the working set of tickets) in CueT's design.

Our approach builds on nearest neighbor classification. In particular, CueT provides triage recommendations for an incoming alarm via a nearest-neighbor strategy that orders the working set of tickets by their similarity to that alarm. Similarity is measured using a distance function that can adapt based on operator actions.

Although the nearest neighbor strategy can help match incoming alarms with existing tickets, there is no easy way to identify that a new ticket needs to be spawned. We extend the classification module to include a mechanism for recommending when an incoming alarm should spawn a new ticket and include this recommendation in CueT's ordered list of recommendations.

New tickets are interactively spawned and added to the working set when a human operator judges that an incoming alarm is unrelated to any existing ticket (i.e., is part of a new problem). Additionally, old tickets are dynamically discarded from the working set over time, simulating the effect of problems represented by those tickets as being resolved. This also serves to reduce interference among existing tickets in the working set. Therefore, an operator has control over both *Input* examples (i.e., alarms) and *Classes* (i.e., tickets) in CueT's interactive machine learning.

Recommending Existing Tickets

CueT measures similarity between the incoming alarm and a ticket in the working set of tickets by computing the average distance between the incoming alarm and each alarm in the ticket. CueT then orders the tickets by their average distance to the incoming alarm and uses this ordering of tickets in its display of triage recommendations (discussed in Section 6.3.2).

Distance between alarms is measured using 17 individual distance metrics, each of which represents alarm similarity along a feature attribute. These 17 distance metrics were found to be relevant and

important for triage during our initial observations of Tier 1 operators (see Section 6.1). The operators typically inspect the following alarm attributes:

- *Device Name* (e.g., ab1-cd2-ef3-gh4)
- *Device Type* (e.g., Router, Switch)
- *Element Type* (device part needing attention, e.g., Port)
- *Name* (includes Device Name and additional information such as the Element that needs attention, e.g., Port-ab1-cd2-ef3-gh4)
- *Severity* (an integer ranging from 1 to 5 representing highest to lowest priority, respectively)
- *Event Name* (e.g., Fan-3-Failed, High Utilization)

From these attributes, CueT computes 17 string-based distance metrics described below. Our simulations described below show that these string-based distance metrics effectively capture an operator's practice of visually comparing the attribute values of alarms. Further, because large organizations often follow standard device naming conventions (e.g., the "ab1" in ab1-cd2-ef3-gh4 typically indicates the location of the device (Spring *et al*, 2002)), some of our string-based metrics implicitly encode topological information about the underlying network structure (e.g., device ab1* is likely to be near device ab2*). If organizations do not include topological information in device names, such information is typically available in network configuration data, from where it could be easily extracted and used to compute the following metrics.

For alarm attributes *Device Name*, *Name*, and *Event Name*, as well as the four standard component parts of the *Device Name* (e.g., ab1-cd2-ef3-gh4 is divided into ab1, cd2, ef3, and gh4), CueT computes two string-based distance metrics. These include the edit distance and the longest common substring (LCS) converted to a distance according to:

$$d_{i,j} = \text{maxlength}\,(i,j) - s_{i,j}$$

where $s_{i,j}$ is the length of the longest common substring between strings i and j. This amounts to fourteen metrics in total.

We include both edit distance and LCS because they have complementary strengths. For example, LCS is a good measure for strings that encode location. Devices "ab1*" and "ac1*" are likely in different

locations. For these, LCS distance (which is 2) better captures that these are different than edit distance (which is 1). As described below, our method of learning the combination of these individual metrics will reduce the effect of any irrelevant metric (edit distance in this case).

For alarm attributes *Device Type*, *Element Type*, and *Severity*, CueT computes one string matching distance metric each (amounting to three metrics in total). This distance metric returns 0 if the attribute values are the same or 1 if they are different.

We combine these 17 distance metrics using Mahalanobis distance, which parameterizes distance between any alarms $\boldsymbol{u}$ and $\boldsymbol{v}$, represented as d dimensional feature vectors, by a $d \times d$ positive definite covariance matrix A:

$$Distance\ (\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u} - \boldsymbol{v})^{T} A\ (\boldsymbol{u} - \boldsymbol{v})$$

This function effectively weights the 17 distances by the matrix A, which encodes their relative importance for alarm classification and the correlations between them.

We learn the parameters of the matrix A from operators using an online metric learning algorithm, originally derived for nearest neighbor classification in static environments (Jain *et al*, 2008). We extend the procedure to dynamic scenarios where both the number and type of classes are varying. In particular, given a stream of alarms, each labeled with the ticket that it was triaged into, we incrementally update the matrix A by encoding the labels as constraints indicating that the incoming alarm and each alarm in the target ticket should be near each other. When an alarm spawns a new ticket, no update is made to the matrix A (however, this does change the working set of tickets). To learn the parameters of A, we initialize it to the identity matrix (and set the regularization parameter η to .001) and then update the parameters as we observe triage actions. We continue this process for N alarms, where N is determined empirically from our simulation experiments described below, and then fix the distance function. The final covariance matrix A_N is used in making recommendations for the remaining data.

Intuitively, the parameters learned for the matrix A reflect the importance and correlations among the individual distance metrics best explaining the human operator's actions. The advantage of learning the matrix A from data is that it does not require expert tuning, which can be difficult to obtain and does not evolve with the network (Gardner and Harle, 1996; Steinder and Sethi, 2004).

Recommending Starting a New Ticket

CueT maintains a threshold distance for starting a new ticket based on information about when operators spawn new tickets. When an operator spawns a new ticket for an incoming alarm, the distance between

this alarm and the nearest ticket in the working set is stored. We experimented with several strategies for computing the threshold distance from these stored distances. They include taking the minimum and average over various window sizes of the most recently stored distances (including a window of all the distances). We found that taking the minimum within a window of the five most recent stored distances performs best. That a small window size performs better than larger sizes likely stems from the fact that thresholds need to reflect the dynamically changing distribution of tickets in the metric space.

For each incoming alarm, CueT computes the latest threshold distance using the strategy above and inserts a "start new ticket" recommendation into its ordered list of recommendations according to this distance.

Spawning New Tickets and Discarding Old Tickets

When an operator determines that an incoming alarm is part of a new problem, a new ticket is created and added to the working set. CueT also automatically discards old tickets, simulating the resolution of problems. We use a windowing mechanism to discard old tickets. In particular, we fix the window size to N, which is the number of alarms used for learning our covariance matrix. Therefore, any time the number of unique alarms in the working set of tickets exceeds N, we remove the oldest ticket in the set. Spawning new tickets and discarding old ones means that the working set of tickets used for machine learning based recommendations is continually evolving as an operator interacts with CueT.

Simulation Experiments

In this section, we present the results of experiments that simulate human interaction with CueT's interactive machine learning component. For our experiments, we obtained alarm triage data from a network operations center at a large organization that monitors a network with approximately 15,000 devices. This data was labeled by Tier 1 operators through their manual triage process. To evaluate CueT's interactive machine learning component over a long period, we use data spread across several months. In particular, we use data from the first day of each month between January and August 2010 (inclusive) except for May and July when the network had recording problems. This data set contains 338,218 alarms of which 8,719 are unique and are mapped to 1,281 unique tickets.

To simulate human interaction and compute the accuracy of CueT's learning, we processed alarms in the data in the order in which they occurred. For each alarm, we first compute an ordered list of recommendations that we use to measure accuracy. Then, we obtain the actual label for the alarm and either add the alarm to an existing ticket or create a new ticket. If we add the incoming alarm to an existing ticket and we have observed fewer than N alarms, we update A_N as described previously. If we

start a new ticket, we update the threshold distance for starting new tickets and update the working set of tickets. Finally, if we determine that the working set has exceeded the window size of N alarms, we discard the oldest ticket in the set.

We measure recommendation accuracy for each incoming alarm by comparing the recommendations to the ground truth (all of the alarm triage data observed before reaching the current incoming alarm, without discarding any tickets). There are two types of correct recommendations:

- *Correct New Ticket*. CueT recommends a new ticket be created and the alarm's label shows that a human operator actually created a new ticket for that alarm.

- *Correct Existing Ticket*. CueT recommends an existing ticket and that ticket is the alarm's actual label.

If neither of these is true, we record the recommendation as incorrect. Note that because the nature of the problem we are dealing with requires that we operate in a moving window, some of our errors may be the result of discarding tickets (e.g., recommending a new ticket when the correct ticket is in the ground truth but no longer exists in the working set of tickets).

Because multiple tickets may be the same distance away from an incoming alarm, we compute recommendation accuracy as whether or not the alarm's actual label appeared within the set of ticket recommendations a given distance away from the alarm or closer. For example, if CueT predicts two different ticket recommendations as being equally closest to the incoming alarm ("Top 1 distance" away) and if the correct label is one of the two tickets then we consider this a correct recommendation at the Top 1 distance. Similarly, a recommendation is correct at the Top 2 distance if the correct label is within the set of tickets recommended at the Top 2 distance away from the incoming alarm or less (e.g., at the Top 1 distance). We experimented with accuracy within the Top 1, 2, 3 and 4 distances from the incoming alarm.

We ran ten simulations over our data, varying the number of alarms N used in both learning the distance function parameters A_N as well as in the window size for discarding old tickets. Figure 6.2 (left) illustrates CueT's accuracy within the set of tickets recommended at the Top 1, 2, 3, and 4 distances from incoming alarms averaged over all the simulation trials. Figure 6.2 (middle and right) also shows the average number and percentage of tickets (out of N) presented at each of the Top 1 to 4 distances. From these results it appears that presenting tickets in the Top 3 distances achieves a good balance between relatively high accuracy and a small number of tickets being presented. Therefore, for our user study described

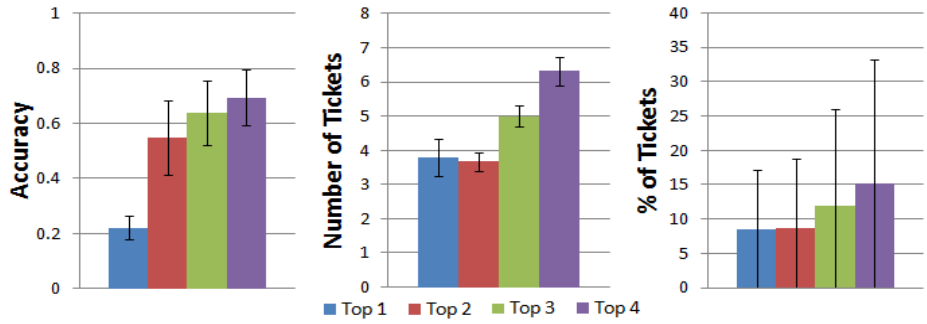

Figure 6.2: Results from our simulated experiments evaluating CueT's interactive machine learning component. On the left is CueT's accuracy within the set of tickets recommended within the Top 1, 2, 3, and 4 distances from each incoming alarm, averaged over all simulation trials. The middle and right-most graphs show the average number and percentage of tickets (out of *N*) presented at each distance, respectively. These results show that presenting tickets within the Top 3 distances achieves a good balance between high accuracy and a small number of presented tickets.

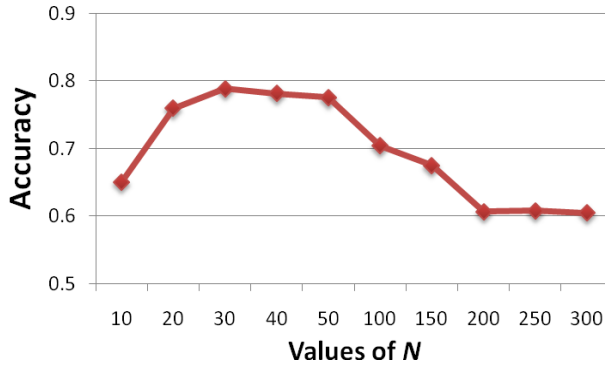

Figure 6.3: This figure displays CueT's accuracy (at the Top 3 distances) during our simulated experiments for various window sizes *N*. From this graph we see that CueT's accuracy peaks at a window size of 30. Therefore, we set *N*=30 in our subsequent user study with real network operators.

later, we fix CueT to recommend tickets for an incoming alarm within the Top 3 distances from that alarm.

CueT's accuracy at the Top 3 distances over the various values of *N* that we experimented with (10, 20, 30, 40, 50, 100, 150, 200, 250, and 300) appears to peak at an *N* value of 30 alarms (see Figure 6.3). Therefore, for our user study, we set *N*=30 in our interactive machine learning engine.

6.3.2 Visualizing Recommendations in CueT's Interface

As illustrated in Figure 6.4, CueT's interface consists of two main views: the *Alarm View* on the right displays alarms as they stream in from the network and the *Ticket View* on the left displays CueT's ticket recommendations.

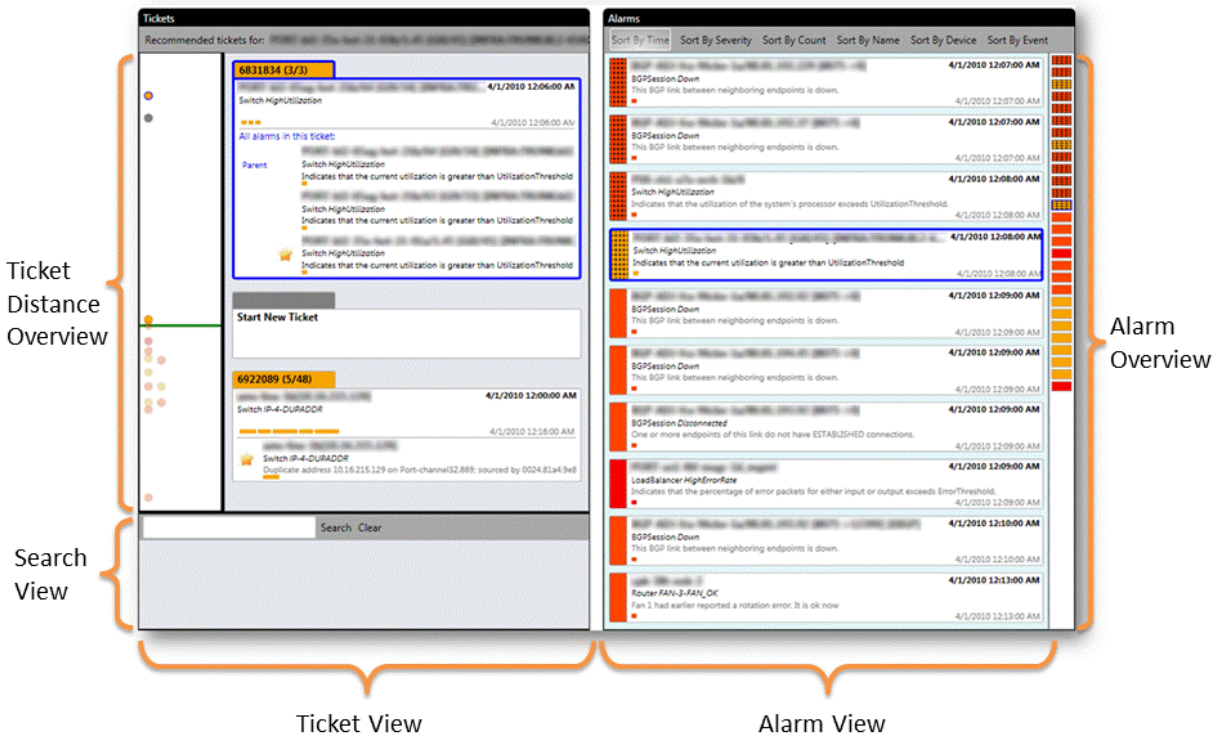

Figure 6.4: CueT uses interactive machine learning to help operators triage alarms streaming in from a computer network. This figure shows CueT's interface. Alarms streaming in from a computer network are displayed on the right. CueT's triage recommendations for each alarm appear on the left along with a visualization of CueT's confidence in those recommendations (far left). Device names and other alarm information are blurred for security reasons.

When operators click on an alarm in the Alarm View, CueT displays its recommendations for that alarm in the Ticket View. Because triage accuracy is a critical *Performance Requirement* of CueT, we designed CueT to include an illustration of its confidence in its triage recommendations, which it displays in its *Ticket Distance Overview* visualization. Operators can then inspect the recommendations and visualizations to determine how to triage the alarm (i.e., by either adding it to an existing ticket or starting a new ticket). Operators triage the alarm by dragging and dropping it onto the appropriate ticket in the Ticket View.

CueT's interface also contains a *Search View* (bottom left in Figure 6.4) through which operators can search for existing tickets by entering a search string as they do with their current system for triaging alarms. Operators can add alarms to tickets appearing in the Search View just as in the Ticket View. This is again important for triage accuracy as it allows an operator to use traditional means for triaging an alarm if they disagree with CueT's recommendations.

Alarms and Alarm View

Our initial observations of operators helped us determine the importance of certain alarm attributes. *Severity* and *Notification Time*, most viewed by operators when determining which alarm to triage next, are displayed prominently. As shown in Figure 6.4 (right), Severity (on a scale of 1-5) is encoded on the left of an alarm by color: red (most severe), orange-red, orange, yellow, and white (least severe). The Notification Time attribute is emphasized in bold and displayed at the top right-hand side of the alarm.

Operators often work under a service level agreement that defines the time limit within which an alarm must be triaged. CueT highlights when an alarm has passed this limit by overlaying a pattern on the Severity display as in the alarms at the top of the Alarm View in Figure 6.4.

Operators also sometimes use the *Count* attribute of an alarm to determine which alarm to deal with next. The Count attribute represents the number of duplicate alarms observed and many duplicates sometimes signal a severe problem. CueT represents Count by the length of the horizontal bar at the bottom of the alarm (color coded by Severity of the alarm). In addition, if duplicates are observed, the time when the most recent duplicate was observed is displayed in gray at the bottom right of the alarm.

Alarm *Name* is also emphasized in the alarm in bold as this attribute is used most often when comparing alarms to existing tickets and deciding how to triage. The rest of the alarm information, including *Device Type*, *Event Name*, and a description of the event is displayed less saliently in the alarm in gray. This layout allows operators to visually scan and compare alarms by Severity, Time, and Count while still being able to digest the rest of the alarm information in a compact representation.

Alarms are displayed in the Alarm View as they stream in from the network. Because operators often miss important alarms that appear off of the screen, the Alarm View includes an Alarm Overview (far right in Figure 6.4) that provides awareness of all alarms still requiring triage even if they are off the screen. This overview displays one rectangle per alarm, color coded by that alarm's Severity and possibly the pattern if that alarm has passed its service level agreement. The heights of the rectangles automatically adjust so as to always display all of the alarms currently available for triage. The rectangles are presented in the order that alarms are displayed in the Alarm View, where alarms can be sorted by any attribute of the alarm. This allows the overview to act as a scroll bar for easy alarm navigation.

Tickets and Ticket View

Tickets are a collection of related alarms (left in Figure 6.4). Each ticket has a parent alarm, which is manually determined by a human operator and typically represents either the most severe or the first alarm in the ticket. The label at the top of the ticket is color coded by Severity of the ticket's parent alarm.

The ticket label also includes the ticket's unique ID, automatically generated by the system at the time of creation. In addition, the ticket label displays two numbers in parenthesis: the number of unique alarms within that ticket followed by the total number of duplicates across those alarms. Unique alarms and duplicates are dually represented in the ticket as a series of horizontal bars (similar to the Count display in the alarm representation). Each bar corresponds to a unique alarm and the length of the bar reflects the number of duplicates for that alarm. Each bar is also color coded by Severity of the corresponding alarm.

Immediately below the ticket label is information about the ticket's parent alarm along with the ticket description. Below the parent alarm is the best matching alarm within the ticket to the incoming alarm (next to the star icons in Figure 6.4). This serves as an explanation for why CueT is recommending that an operator triage an alarm into a given ticket (i.e., it conveys some aspect of CueT's underlying *Logic*, namely the alarm most similar to the incoming alarm in a given ticket). Thus, ticket representations displayed in the Ticket View are tailored for each alarm. Operators can also click on a ticket to display all the alarms currently grouped within the ticket. New ticket recommendations are displayed as empty ticket stubs with a gray label and text within the ticket displaying "Start New Ticket."

Each time the operator clicks on an alarm to triage in the Alarm View, CueT generates its ticket recommendations for the selected alarm and displays them in the Ticket View. They are shown in order of increasing distance from the alarm as per the distance function described previously. By default, and as determined by our simulation results, the Ticket View initially displays only the tickets within the Top 3 distances from the alarm being triaged. This helps to balance operator load and the probability of these tickets containing the correct recommendation. CueT allows operators to reveal more tickets to inspect using the Ticket Distance Overview visualization.

Displaying Quality to Reduce Operator Error

The Ticket Distance Overview (far left in Figure 6.4) is designed to provide operators with an estimate of its confidence in its recommendations so that operators can determine the necessity of inspecting more tickets. Each bubble in the Ticket Distance Overview corresponds to a ticket. The vertical position of the bubbles relative to the top of the overview reflects the distance between the alarm being triaged and each ticket within the working set. That is, the closer the bubble is to the top, the better a match the corresponding ticket is for the alarm with respect to our distance function. Also vertical positions of the bubbles correspond to the ordered list of recommendations in the Ticket View. Vertical distances are normalized so as to fit all of the existing tickets within the display. Thus, the positions of the bubbles display relative distances between tickets rather than absolute distances. Horizontal positioning is only used to minimize overlap of bubbles that are of equal or near equal distance to the incoming alarm.

Bubbles that are positioned near each other are comparable in terms of their similarity to the alarm currently being triaged. In this case, the overview should encourage operators to inspect all of the comparable tickets, important for accurate triage (i.e., for CueT's *Performance Requirements*). To enable operators to inspect more tickets when necessary, CueT provides a horizontal green bar allowing them to set the distance threshold for the tickets to be displayed in the Ticket View. It divides the bubbles into a visible region (above the bar and corresponding to visible tickets in the Ticket View) and the invisible region (faded bubbles below the bar and corresponding to tickets not currently visible in the Ticket View). Operators can drag this bar vertically to reveal and inspect other tickets within the Ticket View.

6.4 Evaluation

We conducted an evaluation to examine the effectiveness of CueT for alarm triage as compared to the traditional method of manually ticketing alarms. For the Traditional condition, we replicated the commercial system used by our network operators (see Figure 6.1). As in the commercial system, participants could keyword search the tabular view of existing tickets to add incoming alarms or create new tickets. Adding alarms to existing tickets or creating new tickets is achieved by right clicking on a row in the table of alarms and selecting the corresponding action from a popup menu. The commercial system has separate windows for displaying alarms and searchable tickets, both using a tabular format. To avoid the overhead of switching between windows (a problem we observed during our initial observations), our version of the Traditional interface combines both of these views in one window as in CueT. This combination provides a fairer evaluation of the current practice, as the two-window issue is easy to fix.

Data

For our study we used part of the data that we used for our simulation experiments (January 1, 2010 data). To compare two interfaces, we created two data sets from this data. To ensure that each set includes comparable numbers, distributions and types of alarms and tickets, we extracted alternating unique alarms. We also simulated the correct assignment of alarms to tickets for alarms not being shown, as CueT relies on a dynamically changing working set of tickets. Therefore, *Data Set 1* included odd alarms (and we simulated the correct assignment for even alarms) and *Data Set 2* included even alarms (and we simulated the correct assignment for odd alarms). Each data set contained 80 unique alarms to be triaged by our participants, as our pilot study showed that was a manageable number of alarms to triage in about 20 minutes. For demonstration and practice in each condition, we also created two additional data sets (*Demo Set 1* and *Demo Set 2*) from the April 1, 2010 data using this same approach.

Study Design and Procedure

We conducted a within-subjects study, with each participant performing alarm triage using both CueT and the Traditional method. We compared *CueT* to the *Traditional* method in terms of accuracy, speed, and user preference. To avoid learning, fatigue, or other carryover effects, we counterbalanced the presentation order of the two interfaces.

We ran participants individually or in pairs depending on their schedule. Each participant worked on a 2.7 GHz dual-core Windows 7 laptop with 4 GB RAM. We attached a 20.1'' Samsung monitor at a resolution of 1200x1600 (i.e., in a portrait orientation) to each laptop, as well as a mouse and keyboard. We turned the laptop away from participants so they could only look at the attached monitor and use the attached mouse and keyboard. When we ran pairs, we faced their desks away from each other to minimize disturbance.

Before each condition, the experimenter demonstrated each interface using *Demo Set 1*. Then the participants were allowed to practice triaging alarms using *Demo Set 2* until they were comfortable with the interface and had practiced triaging several alarms, for a maximum of 5 minutes.

In each condition, participants were asked to triage all 80 presented alarms as accurately and quickly as possible to simulate an actual usage scenario. We also asked them to triage alarms in the order that they normally would (i.e., they could triage the alarms in any order, but are encouraged to triage high severity alarms and alarms that have passed their service level agreement first). All interface actions were time-stamped and logged.

After each condition, participants were given a short questionnaire about the interface they just used. The questionnaire included 7-point Likert scale questions asking participants for their level of agreement with statements about the interface used (e.g., "Overall, I am satisfied with this system.") and specific questions about the CueT interface if they had just used that interface (e.g., "The ticket recommendations were useful."). It also asked participants to list three things that they liked and three that they would like improved about the interface. At the end of the session, a final questionnaire asked participants to select which interface they preferred and explain why.

The experiment lasted about 90 minutes and participants were given a gratuity of $20 worth of dining coupons. To encourage participants to triage alarms quickly and accurately, we also offered a prize of an additional $20 worth of dining coupons for the person who performed the best in terms of speed and accuracy in each condition.

Participants

We recruited eleven people (two female, ages 28 to 44) plus one male pilot from the network operations team. Our participants were not currently working as Tier 1 operators, though six of them were self-proclaimed experts at the alarm triage process and four said they were proficient. One said he was a beginner. We could not recruit active Tier 1 operators because of their tight work schedule and because many work outside of the country.

6.5 Results

Performance: Accuracy and Speed

We analyze our logged data in terms of accuracy and speed. *Accuracy* is computed as the percentage of alarms correctly triaged out of the total presented. Correctness of participant labels is measured against the ground truth labels.

We compute two measures for speed: *Time on Screen* and *Time to Ticket*. The former is the time between when an alarm appeared on screen and when the participant completed the triage operation for that alarm. Along with *Accuracy*, it is a key measure of triage performance and is used to formulate service level agreements that the monitoring team offers. For instance, a possible guarantee may be that for 95% of alarms *Time on Screen* would be under 5 minutes. Note that both *CueT* and *Traditional* present multiple alarms on screen simultaneously and operators need not triage alarms in the order in which they appear on screen. Therefore, *Time on Screen* is affected by the order in which an operator decides to triage an alarm. Thus, for a detailed view of triage behavior, we also study *Time to Ticket*, which is time between successive triage actions regardless of order.

For *Accuracy*, *Time on Screen*, and *Time to Ticket*, we perform paired-samples *t* tests. We report means and standard deviations throughout.

Our analyses showed that participants were able to triage alarms faster with *CueT* than with the *Traditional* interface while maintaining the same level of accuracy. Participants were significantly faster with *CueT* than *Traditional* in terms of *Time on Screen* (M=107.7s, SD=127.7s vs. M=277.8s, SD=168.5s, $t(10)=4.43$, $p=.001$) and in terms of *Time to Ticket* (M=10.1s, SD=2.69s vs. M=12.9s, SD=3.79s, $t(10)=3.26$, $p=.009$). There was no significant difference in terms of accuracy between the *CueT* and *Traditional* conditions (M=71.8%, SD=17% vs. M=76.4%, SD=8%).

Our data included a type of alarm that required special handling. Operators are usually instructed to always create a new ticket for each such alarm, regardless of similarity to other alarms. Only a few participants asked us how to triage such alarms, to which we responded that they should triage as normal.

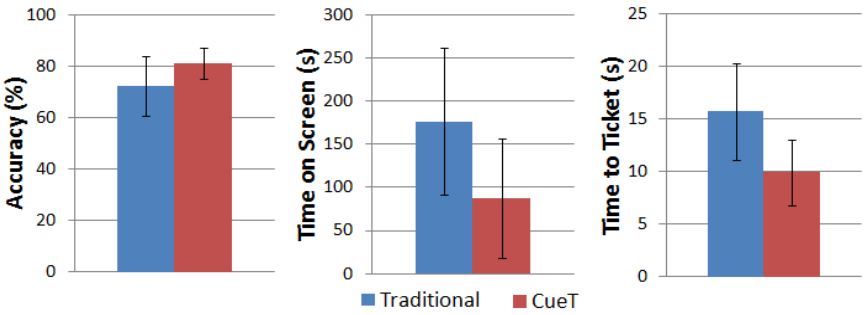

Figure 6.5: Results from our user study comparing CueT to the Traditional state-of-the-art approach for alarm triage. This figure displays Accuracy (left), Time on Screen (middle), and Time to Ticket (right) comparisons. All differences shown here are significant. Error bars represent standard error. From these graphs we see that CueT outperformed the Traditional method in terms of both accuracy and speed.

The logged data shows that such alarms were handled unevenly by participants. Some rapidly created new tickets for these alarms without inspecting recommendations (in CueT) or searching for related tickets (in the CueT or Traditional condition). Others triaged these alarms based on similarity. These alarms reduce CueT's accuracy because its model does not handle exceptional cases. Despite this, our results show that CueT's accuracy is no worse and its speed is much better.

We also re-did our analysis after removing 39 of these exceptional alarms from the data to evaluate CueT's performance in the absence of special cases. Our corrected analyses of variance show that participants were still faster with *CueT* but also more accurate than the *Traditional* condition (see Figure 6.5). The accuracy with *CueT* versus the *Traditional* condition was 81.3% (*SD*=6%) vs. 72.4% (*SD*=12%), ($t(10)=2.29$, $p=.045$). The participants were significantly faster with *CueT* at triaging in terms of both *Time on Screen* (*M*=86.9s, *SD*=69.2s vs. *M*=176.2s, *SD*=85.2s, $t(10)=4.63$, $p=.001$) and *Time to Ticket* (*M*=9.9s, *SD*=3.1s vs. *M*=15.7s, *SD*=4.6s, $t(10)=6.52$, $p<.001$).

Subjective Preference

We analyze our post-condition questionnaires using Friedman Chi-Square tests. *CueT* was favored significantly more than *Traditional* in terms of overall satisfaction ($\chi^2(1,N=11)=9.0$, $p=.003$), how much participants liked using the system ($\chi^2(1,N=11)=11.0$, $p=.001$), whether they felt that they could efficiently ticket alarms with the system ($\chi^2(1,N=11)=6.4$, $p=.011$), and whether they felt the system was easy to use ($\chi^2(1,N=11)=11.0$, $p=.001$) (Figure 6.6).

Regarding *CueT*-specific features, participants tended to agree with the statements "The ticket recommendations were useful" (5.81 avg.) and "The Distance Overview was useful" (5.36 avg.). In addition, all of our participants chose *CueT* as their preferred system for network alarm triage.

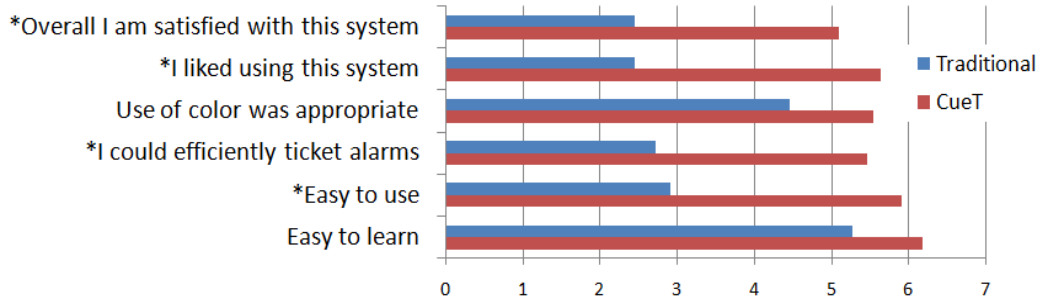

Figure 6.6: Questionnaire results from our user study comparing CueT to the Traditional approach. Questions were 7-point Likert scale. Questions with *'s indicate the difference between CueT and Traditional were significant. These results show that CueT was favored by network operators on all subjective questions.

6.6 Discussion and Future Work

Our results show that real network operators can triage alarms significantly faster with CueT than with their traditional method. When considering general alarms as well as exceptional cases, CueT reduces the *Time on Screen* of alarms by 61.2% on average. When excluding exceptional cases, the savings on *Time on Screen* are reduced to 50.7%. Savings are lower without exceptional cases because such cases require an action that can be performed quickly and without deliberation (e.g., rapidly creating new tickets without looking for similar tickets).

Time on Screen is also affected by the order in which an operator triages alarms. Thus, CueT's Alarm Overview, designed to facilitate awareness of alarms remaining in the queue of alarms to be triaged, likely contributed to the *Time on Screen* savings. In listing the things they liked about CueT, one participant commented that "*Having the alert change colors as it approached [service level agreement] helps with prioritization*."

For *Time to Ticket*, CueT enables a savings of 21.6% when considering general and exceptional alarms, and a savings of 36.8% when considering only the general alarms for which CueT is designed to handle. Participants relied heavily on CueT's recommendations to triage alarms, performing only 4.6 manual searches on average throughout their sessions with CueT (compared to 84.6 searches on average with the traditional method). Only one participant ever ticketed an alarm via dragging and dropping the alarm onto CueT's Search View. Therefore, as *Time to Ticket* measures the time between ticket actions, savings in *Time to Ticket* can be attributed to CueT's ability to provide operators with appropriate suggestions about how to ticket each alarm. To put this in perspective, assuming 10K alarms per day and a time savings of 5.8s per alarm (36.8%), CueT's estimated cumulative time savings using this measure amounts to about 20 operator days per month.

As our observations of operators revealed, exceptional cases are a reality in network operations. CueT currently cannot automatically exclude exceptional cases from its dynamically changing model. Remarkably, when considering general alarms along with exceptional cases, no significant decrease in overall accuracy is observed. This suggests that CueT does not adversely affect operator ability to address exceptional cases. Furthermore, although including exceptional cases in CueT's dynamic model may cause interference in recommendation accuracy for general alarms, CueT still performed significantly better in terms of accuracy than the traditional method for the general case (81.3% versus 72.4%, respectively). This result points at the robustness of using triage recommendations from human-guided interactive machine learning based models. Furthermore, as interference can negatively affect CueT's recommendation accuracy, it is fair to regard CueT's performance results from our evaluation as a lower bound on its potential for improving alarm triage. It may be possible for network administrators to reduce some of this noise by creating temporary rules to automatically remove exceptional alarms, and we recommend such a facility as future work. Despite the potential for interference, nine of our eleven participants commented that CueT's recommendation ability was one of the things they most liked about the system (e.g., "*Recommendations done by the system for appropriate tickets was very useful*.").

In terms of their subjective preference, all participants preferred CueT over the traditional method. The only questions for which there was no significant difference between CueT and the traditional method were whether the use of color was appropriate and whether the system was easy to learn. As with CueT, the traditional method also makes use of color coding to indicate severity of alarms. CueT however takes an additional step in overlaying information about the operator's service level agreement on the colored area of an alarm. In terms of being easy to learn, our participants were all already familiar with the traditional system of manually triaging alarms. Despite this, they were still able to learn and start using CueT in a matter of minutes.

We believe CueT's tight integration between interactive machine learning and visualization is key to its success. As with other distance-based recommendation systems, multiple tickets in CueT can be equally distant from an alarm. Presenting recommendations as a traditional list would therefore require arbitrarily ordering tickets and would likely mislead operators. To ensure high accuracy, CueT's Ticket Distance Overview visualization was specifically designed to show estimates of recommendation quality and encourage operators to inspect comparable tickets. Further, the coupling between machine learning and visualization makes it easy for operators to provide feedback to the system and keep the model up-to-date. However, additional studies that compare CueT's machine learning with and without visualization would shed more light on the value of this integration. We therefore suggest as future work a longitudinal

investigation of operator confidence in the recommendations and the effects on operator vigilance in carefully inspecting those recommendations with and without visualization.

Also, as mentioned in Section 6.2, we did not consider the fact that several operators could be employed for alarm triage at any given time (i.e., we did not consider the *Collaboration* design factor in this exploration). *Collaboration* has the potential to significantly impact the design of CueT and would be an important area for future research. For example, future work might consider an alarm allocation scheme to appropriately distribute alarms and tickets to different operators based on either alarm similarity or operator expertise. In such a case, the system might also include a technique for reviewing and understanding triage decisions of other operators as this may affect the performance of the system and the ability of operators to triage.

6.7 Summary

In this chapter we presented CueT, a novel system that combines visualizations and interactive machine learning to deal with a highly dynamic environment where the groups of interest are not known a-priori and evolve constantly (i.e., where the *Data Source* is streaming and the *Evolutionary Needs* of the interactive machine learning are significant and ongoing). In addition, CueT operates under strict *Performance Requirements* where speed and accuracy are critical.

We implement CueT in the context of triaging network alarms to assist network operators in the complex task of alarm triage. Our evaluation demonstrates that CueT increases operator accuracy as well as speed compared to the state of the art approach of manual alarm triage. All of our study participants preferred CueT over the traditional method, a sentiment that is reflected in one participant's comment: "*the new system compared to the old is hands down better*." While CueT is designed for triaging alarms, we believe that the lessons learned from our work could extend to other scenarios where humans need to organize continuous streams of data (e.g., email, social network feeds).

Chapter 7
Conclusion

This dissertation aims to advance our understanding of how to design effective end-user interaction with machine learning. To this end, we outlined a design space characterizing the factors impacting the interactive machine learning process (Chapter 3) and presented our own explorations of how to design end-user interactive machine learning in three new scenarios (Chapters 4 to 6). In this chapter, we begin to generalize from our experiences by proposing guidelines for the design of future systems and by identifying open challenges and opportunities for future research. Finally, we conclude with a summary of this research.

7.1 Proposed Guidelines for Future Systems

When an end-user's primary Interaction Focus is on training an accurate or reusable model, we should provide them with a Variety of System Feedback and End-User Control Mechanisms.

In our explorations with end-user interactive machine learning for image classification in CueFlik (Chapter 2), we experimented with several new ways of designing *System Feedback* and *End-User Control*. In particular, we included a *Variety* of complex feedback and control mechanisms (e.g., overview-based example selection, a history of previous models, and revision capabilities) and found that these did not seem to overwhelm our participants during our evaluations. Moreover, our participants used all mechanisms whenever they were available (e.g., participants made use of all three revision mechanisms in our second CueFlik exploration) and complained when control was restricted or lacking in diversity (e.g., *"give more options of photos so filtering them will go faster/easier"* and "*[without revision] it felt a little like typing on a keyboard without a backspace key*").

We hypothesize that our participants were willing and able to attend to and use multiple feedback and control mechanisms because their primary *Interaction Focus* with CueFlik was to train an accurate and reusable model. That is, because a person is expecting to dedicate time and effort to training a model in CueFlik, they were willing to tolerate more complex feedback and desired more control. Conversely,

complex feedback and control mechanisms may be inappropriate when training is a secondary goal or when the model is disposable (e.g., as during interaction with ReGroup as discussed in Chapter 3).

Feedback and Control Variety should be balanced with appropriate levels of Guidance on how to best improve the system.

While people might be willing to tolerate or may desire a *Variety* of feedback and control mechanisms when training accurate and reusable models, we also recommend balancing variety with appropriate levels of *Guidance* on how to effectively use those mechanisms. For example, our explorations with CueFlik (Chapter 2) demonstrated that our overview-based feedback mechanisms guided people to select better examples during training compared to when selection was less restricted. Yet, participants still pursued a range of different training strategies using these overviews. Some participants placed varying emphasis on providing positive or negative examples. Some participants tended to reinforce the system's predictions, while others corrected it. Finally, some participants labeled examples one at a time, while others would select a dozen examples before giving them all the same label. We saw a similar diversity of training strategies in our second CueFlik exploration presenting end-users with a variety of comparison and revision capabilities (e.g., some participants explored models in a depth first manner while others explored using a breadth first approach). We hypothesize that some training strategies are more effective than others for creating accurate models. Therefore, while a *Variety* of feedback and control mechanisms might be beneficial or desired when interaction is a person's primary focus, opportunities remain for identifying effective strategies and steering an end-user towards these while maintaining their sense of control and flexibility.

Concise summaries of model quality can facilitate training of accurate models; however, summaries should not mislead or misrepresent the system's capabilities.

We designed and evaluated several new techniques for summarizing model quality during end-user interactive machine learning in a variety of scenarios. For example, both our overview-based feedback techniques in CueFlik (Chapter 4) and our distance overview in CueT (Chapter 6) contributed to participant ability to assess and guide the machine learning in each scenario. However, the plot of a model's estimated quality in CueFlik's history visualization proved to be distracting to participants in our second CueFlik exploration. Although the plot used an accepted machine learning metric to estimate quality (leave-one out cross validation on the current training data), end-users in our evaluation placed too much trust in the quality score (even when error bars were included). For example, one participant commented *"I wanted the graph to go up instead of concentrating on [the results]"*. Therefore, while machine learning experts might place correct emphasis on such estimates, care must be taken to consider the target user's expectations or mental models of machine learning when designing summaries of model quality.

Light-weight explanations of model quality should be favored over more complex explanations when interaction is not a person's primary focus.

Previous research on explanations for interactive machine learning systems have explored the utility of detailed descriptions or dialog-based explanations to convey system behavior (e.g., Lim *et al*, 2009; Stumpf *et al*, 2007; Tullio *et al*, 2007). Kulesza *et al* (2011) further advocates that explanations should be manipulable and not overly simplistic. However, complex explanations can impede efficiency. Furthermore, while end-users might tolerate complex explanations when interaction with the machine learning is their primary focus, we hypothesize that lighter-weight explanations might be equally effective and preferred when training is not a person's primary task. For example, ReGroup's filter suggestions (Chapter 5) did not demand end-user attention like dialog-based explanations, however, they still served useful for helping our study participant's understand the system's behavior (e.g., "*The filters helped me understand what patterns the tool was discovering, and thus helped understand the tool's behavior"*). Future work might therefore explore other light-weight explanation facilities when designing systems in which training is not a person's primary task.

Interactive machine learning systems should fail gracefully when the underlying algorithm cannot sufficiently capture an end-user's desired concept.

Machine learning systems can rarely perfectly model an end-user's desired concept. When the underlying modeling language is expressive enough to adequately model the true concept, system feedback techniques such as summaries of model quality can help end-users assess and guide the system's understanding. However, further measures should be taken when a desired concept is beyond the capabilities of the system in order to prevent wasted time and effort. For example, we designed ReGroup's machine learning to avoid repeated recommendations of skipped people when the system could not capture the desired group (see Section 5.3.5). Future work might investigate additional techniques for detecting when the system cannot model the intended concept and possibly alerting a user to these situations to minimize frustration. For example, Horvitz (1999) advocated explicitly modeling uncertainty about an end-user's goals in order to scope automated services when uncertain.

Alternatively, when a person's primary focus during interaction is on some domain specific goal (e.g., not the training of an accurate model), alternative methods for achieving those goals should be provided when possible. For example, we provided mechanisms for end-users to manually search for friends by name in ReGroup when the system was not displaying a desired friend (Chapter 5). Similarly in CueT, operators could search for groups via the traditional search interface if appropriate recommendations were not being shown (Chapter 6). Such alternative facilities can enable end-users to progress towards their goals even when adequate modeling by the system is not possible.

7.2 Open Challenges and Opportunities for Future Research

Many challenges and opportunities remain for advancing our understanding of how to design end-user interaction with machine learning systems. For example, all of our explorations examined end-user interaction with machine learning over a large screen provided by a traditional desktop environment. However, as mobile devices become more ubiquitous and computing moves off the desktop, opportunities remain for exploring the design of end-user interactive machine learning in these more restricted *Interaction Environments*. Similarly, understanding how to design end-user interaction with machine learning over very large *Volumes* of data will become increasingly important as big data becomes more prevalent.

The impact of personal *Motivation* (see Section 3.1.2) on an end-user's ability to train effective machine learning systems also warrants further investigation. For example, a person intrinsically motivated to use machine learning for their own purposes (e.g., to create a classifier for their own personal use) might be more inclined to carefully interact with the system (e.g., to inspect and verify a learned model) than a person employed to do so (e.g., our CueT operators discussed in Chapter 7). If this is the case, then new techniques for incentivizing or engaging less motivated users to carefully attend to system feedback or provide accurate guidance should be explored.

Further exploration of the impact of *Machine Learning Experience* on end-user acceptance of machine learning systems and their ability to train effective models is also necessary. This is particularly important as people begin to gain more experience and proficiency in interacting with machine learning systems. For example, we observed in our explorations that people were able to understand certain aspects of the machine learning after interacting with a system for some time (e.g., some participants using ReGroup were able to tell why the system was or was not working in some cases). If such experiences increase end-user empathy towards machine learning systems or enable end-users to more effectively interact with them, then the current emphasis on fostering user trust and acceptance of machine learning systems (discussed in Section 2.2) might be misguided and unnecessary. Instead, focus might turn to designing interaction differently for varying levels of expertise.

Important opportunities also remain for scaling up the impact and usability of data by leveraging groups or crowds of people to collaboratively drive interactive machine learning systems. For example, our exploration with CueT (Chapter 6) assumed one operator would be triaging alarms at any given time. However, most network operations centers employ multiple operators to triage alarms. Therefore, examining the *Collaboration* design factor, particularly in understanding how to coordinate the efforts of multiple people interacting with a machine learning system, is an open and important area of future work.

Furthermore, as interactive machine learning becomes more prevalent in our everyday applications, people should be able to share and re-use machine learned models rather than starting from scratch. For example, imagine an online repository where people can post, tag, rate, and search for CueFlik trained image classifiers. Alternatively, imagine a browser plug-in that automatically presents a person with potentially useful end-user created classifiers in the context of search, social media, or other data-intensive multimedia applications, allowing a person to choose to apply a classifier based on its rating and description. People should also be able to bootstrap, build upon, and combine previous models to configure more sophisticated data processing and automated behaviors. To achieve these possibilities, we must understand how people can meaningfully describe, compare, and search for existing machine learning models, how models can be generalized or transformed for new situations and purposes, and how we can create composable models to enable more powerful automation.

Finally, while in the previous section we began to hypothesize about guidelines for designing future systems, future work should further progress towards a more prescriptive design space. For example, future work might examine similar design solutions (e.g., such as those identified in Section 3.2) in different scenarios defined by our design factors outlined in Section 3.1. For instance, CueFlik's feedback and control capabilities should be evaluated in a system in which interactive machine learning is not a person's primary goal, in order to see how receptive people are to these levels of feedback and guidance in different scenarios. Similarly, ReGroup's feature-based feedback technique should be evaluated for different types of data. Such research can further advance our understanding of how to design effective end-user interaction with machine learning.

7.3 Conclusion

Machine learning is the key to unlocking powerful information contained in the vast amount of data we now have available to us. However, the complexity of machine learning has largely restricted its application and use to experts and skilled developers. For example, trained developers can employ machine learning over data to automatically detect objects of interest, organize information, and understand and predict behaviors. In contrast, the most a typical person can currently do with large amounts of data is search over it.

This dissertation examines the fundamental process by which everyday people can interact with machine learning, known as end-user interactive machine learning. While most recent work has focused on demonstrating the utility of end-user interactive machine learning in specific applications, this dissertation investigates how the end-user interactive machine learning process should be designed. In particular, this dissertation pushes the boundaries of what end-users can do with machine learning by designing new techniques and systems involving end-user interactive machine learning in three new application scenarios.

In addition, this dissertation characterizes general design factors that impact and constrain an end-user's interaction with machine learning-based systems. Together, these contributions move us beyond ad-hoc designs for specific applications and provide a foundation for future researchers and developers of end-user interactive machine learning systems.

REFERENCES

1. Aberdeen, D., Pacovsky, O. and Slater, A. (2010) The Learning Behind Gmail Priority Inbox. *NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds*.
2. Abowd, G., Atkenson, C., Hong, J., Long, S., Kooper, R. and Pinkerton, M. (1997) CyberGuide: A Mobile Context-Aware Tour Guide. *Wireless Networks – Special Issue: Mobile Computing and Networking 3* (5): pp. 421-433.
3. Adomavicius, G. and Tuzhilin, A. (2005) Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering 17* (6): pp. 734-749.
4. Agichtein, E., Brill, E. and Dumais, S. (2006) Improving Web Search Ranking by Incorporating User Behavior Information. *SIGIR 2006*, pp. 19-26.
5. Alpert S.R., Karat J., Karat C., Brodie C. and Vergo J.G. (2003) User attitudes regarding a user-adaptive e-Commerce web site. *UMUAI 13* (4): pp. 373–396.
6. Amershi, S., Fogarty, J., Kapoor, A. and Tan, D. (2010) Examining Multiple Potential Models in End-User Interactive Concept Learning. *CHI 2010*, pp. 1357-1360.
7. Amershi, S., Fogarty, J., Kapoor, A. and Tan, D. (2009) Overview-Based Examples Selection in Mixed-Initiative Interactive Concept Learning. *UIST 2009*, pp. 247-256.
8. Amershi, S., Fogarty, J. and Weld, D. (2012) ReGroup: Interactive Machine Learning in Social Network Access Control. *CHI 2012*, pp. 21-30.
9. Amershi, S., Lee, B., Kapoor, A., Mahajan, R. and Christian, B. (2011) CueT: Human-Guided Fast and Accurate Network Alarm Triage. *CHI 2011*, pp. 157-166.
10. Amoo, T. and H. H. Friedman. (2001) Do numeric values influence subjects' responses to rating scales? *Journal of International Marketing and Marketing Research, 26* (41–46), February 2001.
11. Anderson, J. and Rainie, L. (2012) The Future of Big Data. *Report of the Pew Internet & American Life Project,* July 20, 2012.
12. Appleby, K., Goldszmidt, G., and Steinder, M. (2001) Layered Event Correlation Engine for Multi-Domain Server Farms. *IEEE INM 2001,* pp. 329-344.
13. Bacon, K. and Dewan, P. (2011) Mixed-Initiative Friend-List Creation. *ECSCW 2011*, pp. 293-312.
14. Baily, B.P., Konstan, J.A. and Carlis, J. (2001) The Effects of Interruptions on Task Performance, Annoyance, and Anxiety in the User Interface. *INTERACT 2001*, pp. 593-601.
15. Baker, K., Bhandari, A., and Thotakura, R. (2009) An Interactive Automatic Document Classification Prototype. *HCIR 2009 Workshop on Human -Computer Interaction and Information Retrieval*, pp. 30-33.

16. Baker, R., Corbett, A., Koedinger, K., Evenson, S., Roll, I., Wagner, A., Naim, M., Raspat, J., Baker, D. and Beck, J. (2006) Adapting to When Students Game an Intelligent Tutoring System. *ITS 2006*, pp. 392-401.
17. Barkhuus, L. & Dey, A.K. (2003) Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined. *Ubicomp 2003*, pp. 149–156.
18. Basu, S., Fisher, D., Drucker, S.M., and Lu, H. (2010) Assisting Users with Clustering Tasks by Combining Metric Learning and Classification. *AAAI 2010*.
19. Baum, E.B. and Lang, K. (1992) Query Learning can work Poorly when a Human Oracle is Used. *Neural Networks 1992*.
20. Beckhusen, R. (2012) Study: WikiLeaked Data Can Predict Insurgent Attacks. *Wired.com*, July 17, 2012.
21. Belkin, N., Dumais, S., Scholtz, J. and Wilkinson, R. (2004) Evaluating interactive information retrieval systems: opportunities and challenges. *CHI 2004 Extended Abstracts*, pp. 1594-1595.
22. Bellotti, V. and Edwards, W.K. (2001) Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human-Computer Interaction 16* (2-4): pp. 193-212.
23. Benyon, d. (1993) Adaptive Systems: A Solution to Usability Problems. *Journal of User Modeling and User-Adapted Interaction 3* (1): pp. 1 -22.
24. Bergman, L., Castelli, V., Lau, T. and Oblinger, D. (2005) DocWizards: A System for Authoring Follow-Me Documentation Wizards. *UIST 2005*, pp. 191-200.
25. Bernstein, M., Marcus, A., Karger, D.R. and Miller, R. C. (2010) Enhancing Directed Content Sharing on the Web. *CHI 2010*, pp. 971-980.
26. Billsus, D., Hilbert, D. and Maynes-Aminzade, D. (2005) Improving proactive information systems. *IUI 2005*, pp. 159-166.
27. Billsus, D. and Pazzani, M. (1998) Learning Collaborative Information Filters. *ICML 1998*, pp. 46-54.
28. Billsus, D. and Pazzani, M. (1999) A Personal News Agent That Talks, Learns and Explains. *Agents 1999*, pp. 268-275.
29. Billsus, D. and Pazzani, M. (2000) User Modeling for Adaptive News Access. *UMUAI 10* (2-3): pp. 147-180.
30. Bing, Adapting Search to You. (2011) http://www.bing.com/community/site_blogs/b/search/archive/2011/09/14/adapting-search-to-you.aspx
31. Blum, A., Chalasani, P., Goldman, S.A. and Slonim, D.K. (1995) Learning with Unreliable Boundary Queries. *COLT 1995*, pp. 98-107.
32. Boone, G. (1998) Concept Features in Re:Agent, an Intelligent Email Agent. *Autonomous Agents 1998*, pp. 141-148.
33. Borgman, C. (1986) The User's Mental Model of an Information Retrieval System. *Journal of Man-Machine Studies 24* (1): pp. 47-64.

34. Breen, C. (2011) Identifying faces faster in iPhoto. *Macworld.com*, March 28, 2011.

35. Breese, J. S., Heckerman, D. and Kadie, C. (1998) Empirical Analysis of Predictive Algorithms for Collaborative Filtering. *UAI 1998*, pp. 43-52.

36. Brugnosi, S., Bruno, G., Manione, R., Montariolo, E., Paschetta, E., and Sisto, L. (1993) An Expert System for Real Time Fault Diagnosis of the Italian Telecommunications Network. *IEEE INM 1993*, pp. 617-628.

37. Bunt, A., Conati, C. and McGrenere, J. (2004) A Mixed-Initiative Approach to Interface Personalization. *AI Magazine 30* (4).

38. Byde, A., Wan, H. and Cayzer, S. (2007) Personalized Tag Recommendations via Tagging and Content-based Similarity Metrics. *ICWSM 2007*.

39. Carstensen, J. (2012) Berkeley Group Digs in to Challenge of Making Sense of All that Data. *The New York Times*, April 7, 2012.

40. Cha, A. E. (2012) 'Big data' from social media, elsewhere online redefines trend-watching. *The Washington Post, Business*, June, 6, 1012.

41. Chen, F., Gargi, U., Niles, L., and Schuetze, H. (1999) Multi-Modal Browsing of Images in Web Documents. *SPIE Document Recognition and Retrieval VI 1999,* pp. 122-133.

42. Chen, H.M., Chang, M.H., Chang, P.C., Tien, M.C., Hsu, W.H. and Wu, J.L. (2008) SheepDog: Group and Tag Recommendation for Flikr Photos by Automatic Search-based Learning. Multimedia 2008, ACM Press, pp. 737-740.

43. Chen, J. and Weld, D. (2008) Recovering from Errors during Programming by Demonstration. *IUI 2008*, pp. 159-168.

44. Chen, S. (2010) Can Facebook get you Fired? Playing it Safe in the Social Media World. *CNN*, Nov. 10, 2010. http://articles.cnn.com/2010-11-10/living/facebook.fired.social.media.etiquette_1_social-media-worker-posts-workplace-complaints.

45. Corbett, A. T., Koedinger, K. R. and Anderson, J. R. (1997) Intelligent Tutoring Systems. In M. Helander, T. K. Landauer, and P. V. Prabhu (eds.) *Handbook of Human-Computer Interaction* (2nd ed), Amsterdam: North-Holland, pp. 849–874.

46. Cosley, D. S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl (2003) Is seeing believing?: How recommender system interfaces affect users' opinions. *CHI 2003,* pp. 585–592.

47. Culotta,A., Kristjansson, T., McCallum, A., and Viola P. (2006) Corrective feedback and persistent learning for information extraction. *Journal of Artificial Intelligence 170* (14): October 2006, pp. 1101-112.

48. Cramer H., Evers V., Van Someren, M., Ramlal, S., Rutledge, L., Stash, N., Aroyo, L. and Wielinga, B. (2008) The effects of transparency on trust and acceptance in interaction with a content-based art recommender. *UMUAI 18* (5): pp. 455-496.

49. Cramer, H., V. Evers, M. van Someren, and B. Wielinga. (2009) Awareness, training and trust in interaction with adaptive spam filters. *CHI 2009*, pp. 909–912.

50. Crawford, E., Kay, J. and McCreath, E. (2001) Automatic Induction of Rules for E-mail Classification. *ADCS2001*, pp. 13-20.

51. Crucianu, M., Ferecatu, M. and Boujemaa, N. (2004) Relevance Feedback for Image Retrieval: A Short Survey. In State of the Art in Audiovisual Content-Based Retrieval, Information Universal Access and Interaction, Including Datamodels and Languages. *DELOS2 European Network of Excellence*. INRIA Rocquencourt, France.

52. Cutting, D.R., Karger, D.R., Pedersen, J.O., and Tukey, J.W. (1992) Scatter/Gather: A Cluster-Based Approach to Browsing Large Document Collections. *SIGIR 1992*, pp. 318-329

53. Cypher, A. (1991) Eager: Programming Repetitive Tasks by Example. *CHI 1991*, pp. 33-39.

54. Daneu, L. and Dorizzi, B. (1996) On-line cursive script recognition: a user-adaptive system for word recognition. *Pattern Recognition 29* (12): pp. 1981-1994.

55. Davies, N., Siewiorek, D. and Sukthankar, R. (2008) Activity-Based Computing. *IEEE Pervasive Computing 7* (2): pp. 20-21.

56. Denison, D.C. (2012) Mass. and MIT Launching Big Data Initiatives. *The Boston Globe*, May 30, 2012.

57. desJardins, M., MacGlashan, J., and Ferraioli, J. (2007) Interactive Visual Clustering. *IUI 2007*, pp. 361-364.

58. Dey, A.K., Hamid, R., Beckmann, C., Li, I. and Hsu, D. (2004) a CAPpella: Programming by Demonstrations of Context-Aware Applications. *CHI 2004*, pp. 33-40.

59. Dey, A.K., Mankoff, J., Abowd, G. and Carter, S. (2002) Distributed Mediation of Ambiguous Context in Aware Environments. *UIST 2002*, pp. 121-130.

60. Dix, A., Finlay, J., Abowd, G.D and Beal, R. (2004) Interaction Design Basics. *Ch. 5 in human computer interaction* (3rd ed). Harlow, England: Pearson Education Ltd, pp. 189-224.

61. Domingos, P. and Pazzani, M. (1997) On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning 29*, pp.103-130.

62. Donmez, P., Rose, C., Stegmann, K., Weinberger, A. Fischer, F. (2005) Supporting CSCL with Automatic Corpus Analysis Technology. *CSCL 2005, International Society of the Learning Sciences*, pp. 125-134.

63. Druck, G., Settles, B. and McCallum, A. (2009) Active learning by labeling features. *EMNLP 2009*, pp. 81-90.

64. Drucker, S. M., Fisher, D. and Basu, S. (2011) Helping Users Sort Faster with Adaptive Machine Learning Recommendations. *INTERACT 2011*, pp. 187-203.

65. Duhigg, C. (2012) How Companies Learn Your Secrets. *The New York Times Magazine*. February 16, 2012.

66. Dzindolet, M.T., Peterson, S.A., Pomranky, R.A., and Pierce, L.G. (2003) The role of trust in automation reliance. *International Journal of Human-Computer Studies 58* (6): pp. 697–718.

67. EMC Ionix, http://www.emc.com/products/family/ionix-family.htm

68. Fails, J.A., Olsen Jr., D.R. (2003) Interactive Machine Learning. *IUI 2003*, pp. 39-45.

69. Fang, L. and LeFevre, K.. (2010) Privacy Wizards for Social Networking Sites. *WWW 2010*, pp. 351-360.

70. Fiebrink, R., Trueman, D. and Cook, P.R. (2009) A Meta-Instrument for Interactive, On-the-fly Machine Learning. *NIME 2009*.

71. Fisher, D., Maltz, D.A., Greenberg, A., Wang, X., Warncke, H., Robertson, G., and Czerwinski, M. (2008) Using Visualization to Support Network and Application Management in a Data Center. *IEEE INM 2008*, pp. 1-6.

72. Fogarty, J. and Hudson, S. (2007). Toolkit Support for Developing and Deploying Sensor-Based Statistical Models of Human Situations. *CHI 2007*, pp. 135-144.

73. Fogarty, J., Tan. D., Kapoor, A. and Winder, S. (2008) CueFlik: Interactive Concept Learning in Image Search. *CHI 2008*, pp. 29-38.

74. Frazier, M., Goldman, S., Mishra, N. and Pitt, L. (1996) Learning from a Consistently Ignorant Teacher. *Journal of Computing Systems Science 52* (3): pp. 472-492.

75. Gajos, K., Wobbrock, J.O. and Weld, D.S. (2008) Improving the Performance of Motor-Impaired Users with Automatically Generated, Ability-Based Interfaces. *CHI 2008*, pp. 1257–1266.

76. Gardner, R.D. and Harle, D.A. (1996) Methods and Systems for Alarm Correlation. *GLOBECOM 1996*, pp. 136-140.

77. Garland, A., Ryall, K. and Rich, C. (2001) Learning Hierarchical Task Models by Defining and Refining Examples. *K-CAP 2001,* pp. 44-51.

78. Gass, T., Dreuw,P. Ney. H. (2009) Jointly optimizing relevance and diversity in image retrieval. *CIVR 2009*.

79. Gauch, S., Speretta, M., Chandramouli, A. and Micarelli, A. (2007) User Profiles For Personalized Information Access. In Peter Brusilovsky, Alfred Kobsa and Wolfgag Nejdl (eds.) *The Adaptive Web*, Springer, pp. 54–89.

80. Ghorab, M.R., Zhou, D., O'Connor, A. and Wade, V. (2012) Personalized Information Retrieval: Survey and Classification. *User Modeling and User-Adapted Interaction 1* (63).

81. Gilbert, E. and Karahalios, K. (2009) Predicting Tie Strength with Social Media. *CHI 2009*, pp. 211–220.

82. Ginsberg, J., Mohebbi, M. H., Patel, R. S., Brammer, L., Smolinski, M. S. and Brillian, L. (2009) Detecting Influenza Epidemics Using Search Engine Query Data. *Nature 457*, pp. 1012-1014.

83. Glass, A., McGuinness, D. and Wolverton, M. (2008). Toward Establishing Trust in Adaptive Agents. *IUI 2008*, pp. 227-236.

84. Goo, S. K. (2012) Facebook: a Profile of its 'Friends'. *PewResearchCenter Publications*, May 16, 2012.

85. Goodrich, M. and Schultz, A. (2007) Human Robot Interaction: A Survey. *Foundations and Trends in Human-Computer Interaction 1* (3): pp. 203-275.

86. Google Web History. http://www.google.com/history/.

87. Graham, R. and Caverlee, J. (2008) Exploring Feedback Models in Interactive Tagging. *IEEE/WIC/ACM WI-IAT 2008*, pp. 141-147.

88. Greenberg, S. and Witten, I. (1985) Adaptive Personalized Interfaces: A Question of Viability. *Behavior and Information Technology 4* (1): pp. 31–45.

89. Greengrass, E. (2000) Information Retrieval: A Survey. University of Maryland, Baltimore County.

90. Gross, R. and Acquisti, A. (2005) Information Revelation and Privacy in Online Social Networks. *WPES 2005*, pp. 71-80.

91. Guo, Y. and Schuurmans, D. (2008) Discriminative batch mode active learning. *NIPS 2008*, pp. 593–600

92. Gurevich, N., Markovitch, S. and Rivlin, E. (2006) Active Learning with Near Misses. *AAAI 2006*, pp. 362-367.

93. Hardy, Q. (2012) How Big Data Gets Real. *The New York Times – Technology Bits*, June 4, 2012.

94. Harman, D. (1992) Relevance Feedback and Other Query Modification Techniques. In William B. Frames and Ricardo Baeza-Yates (eds.) *Information Retrieval: Data Structures & Algorithms*, Prentic-Hall, Chapter 11, pp. 241-263.

95. Harrison, D. and Klein, K.J. (2007) What's the difference? Diversity constructs as separation, variety, or disparity in organizations. *Academy of Management Review 32* (4): pp. 1199–1228.

96. Hartmann, B., Abdulla, L., Mittal, M. and Klemmer, S.R. (2007) Authoring Sensor-Based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. *CHI 2007*, pp. 145-154.

97. Hearst, M. (2006) Design Recommendations for Hierarchical Faceted Search Interfaces. *SIGIR Workshop on Faceted Search 2006*.

98. Hearst, M.A., Karger, D.R. and Pederson, J. O. (1995) Scatter/Gather as a Tool for the Navigation of Retrieval Results. *AAAI 1995 Fall Symposium on Knowledge Navigation.*

99. Herlocker, J., Konstan, J., Riedl, J. (2000) Explaining collaborative filtering recommendations. *CSCW 2000*, pp. 241-250.

100. Hoffman, R., Amershi, S., Patel, K., Wu, F., Fogarty, J., Weld, D.S. (2009) Amplifying Community Content Creation with Mixed-Initiative Information Extraction. *CHI 2009*, pp. 1849-1858.

101. Höök, K. (2000) Steps To Take Before Intelligent User Interfaces Become Real. *Interacting with Computers 12* (4).

102. Horvitz, E. (1997) Compelling Intelligent User Interfaces: How Much AI is Enough? *Panel at IUI 1997* (eds.) Johanna Moore, Ernest Edmonds, and Angel Puerta, ACM, Orlando, Florida, 1997.

103. Horvitz, E. (1999) Principles of Mixed-Initiative User Interfaces. *CHI 1999*, pp. 159-166.

104. Horvitz, E., Breese, J., Heckerman, D., Hovel, D. and Rommelse, K. (1998) The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. *UAI 1998*, pp. 256-265.

105. Horvitz, E., Koch, P. and Apacible, J. (2004) BusyBody: Creating and Fielding Personalized Models of the Cost of Interruption. *CSCW 2004*, pp. 507-510.

106. HP OpenView, http://openview.hp.com

107. Jain, P., Kulis, B., Dhillon, I.S., and Grauman, K. (2008) Online Metric Learning and Fast Similarity Search. *NIPS 2008*, pp. 761-768.

108. Jakobson, G. and Weissman, M.D. (1993) Alarm Correlation: Correlating multiple network alarms improves telecommunications network surveillance and fault management. *IEEE Network 7* (6): pp. 52-59.

109. Jameson, A. (2008) Adaptive Interfaces and Agents. In A. Sears and J.A. Jacko (eds.) *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications* (2nd ed.), CRC Press, Boca Raton, FL., pp. 433-458.

110. Jameson, A. (2009) Understanding and Dealing with Usability Side Effects of Intelligent Processing. *AI Magazine 30* (4).

111. Jones, S. and O'Neill, E. (2010) Feasibility of Structural Network Clustering for Group-Based Privacy Control in Social Networks. *SOUPS 2010*.

112. Jøsang, A. and Lo Presti, S. (2004) Analysing the Relationship Between Risk and Trust. *iTrust 2004*, pp. 135-145.

113. Kapoor, A., Horvitz, E., and Basu, S. (2007) Selective Supervision: Guiding Supervised Learning with Decision-Theoretic Active Learning. *IJCAI 2007*.

114. Kapoor, A., Lee, B., Tan, D. and Horvitz, E. (2010) Interactive Optimization for Steering Machine Classification. *CHI 2010*, pp. 1343-1352.

115. Kincaid, J. (2011) Kleiner-Backed Katango Organizes Your Facebook Friends into Groups for You. *TechCrunch.com,* July 11, 2011. http://techcrunch.com/2011/07/11/kleiner-backedkatango-automatically-organizes-your-facebook-friendsinto-groups-for-you

116. Klementtinen, M., Mannila, H., and Toivonen, H. (1999) Rule Discovery in Telecommunication Alarm Data. *Journal of Network and Systems Management 7* (4): pp. 395-423.

117. Kloppel, S., Stonnington, C. M., Chu, C., Draganski, B., Scahill, R. I., Rohrer, J. D., Fox, N. C., Jack Jr, C., Ashburner, J. and Frackowiak, R. (2008) Automatic Classification of MR scans in Alzheimer's Disease. *Oxford Journals – Brain 131* (3): pp. 6810689.

118. Kobsa A., Koenemann J., Pohl, W. (2001) Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships. *The Knowledge Engineering Review 16* (2): pp. 111-155.

119. Koenemann, J. and Belkin, N. J. (1996) A Case for Interaction: A Study of Interactive Information Retrieval Behavior and Effectiveness. *CHI 1996*, pp. 205-212.

120. Kohavi, R. (1995) A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *IJCAI 1995*, pp. 1137-1143.

121. Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L. and Riedl, J. (1997) GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM 40* (3): pp. 77-87.

122. Konstan, J. and Riedl, J. (2012) Recommender Systems: from Algorithms to User Experience. *UMUAI 22*, pp. 101-123.

123. Koren, J., Zhang, Y. and Liu, X. (2008) Personalized Interactive Faceted Search. *WWW 2008*, pp. 477-486.

124. Koren, Y. (2009) Collaborative Filtering with Temporal Dynamics. *KDD 2009*, pp. 447-456.

125. Korkki, P. (2010) Is Your Online Identity Spoiling Your Chances? *New York Times,* Oct. 9, 2010. http://www.nytimes.com/2010/10/10/jobs/10search.html

126. Koychev, I. and Schwab, I. (2000) Adaptation to Drifting User's Interests. *ECML 2000 Workshop on Machine Learning in New Information Age*, pp. 39-46.

127. Krause, A., Singh, A. and Guestrin, C. (2008) Near-optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *Journal of Machine Learning Research 9*, pp. 235-284.

128. Krulwich, B. (1997) LifeStyle Finder: Intelligent User Profiling Using Large-Scale Demographic Data. *AI Magazine 18* (2): pp. 37–45.

129. Kulesza, T., Stumpf, S., Burnett, M. and Kwan, I. (2012) Tell me more? The effects of mental model soundness on personalizing an intelligent agent. *CHI 2012*.

130. Kulesza, T., Stumpf, S., Burnett, M., Wong, W-K., Riche, Y., Moore, T., Oberst, I., Shinsel, A. and McIntosh, K. (2010) Explanatory Debugging: Supporting End-User Debugging of Machine-Learned Programs. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2010), pp. 41–48.

131. Kulesza, T., Stumpf, S., Wong, W.-K., Burnett, M., Perona, S., Ko, A. and Obsert, I. (2011) Why-Oriented End-User Debugging of Naive Bayes Text Classification. *ACM Transactions on Interactive Intelligent Systems 1* (1).

132. Lakkaraju, K, Yurcik, W., and Lee, A.J. (2004) NVisionIP: Network Visualizations of System State for Security Situational Awareness. *VizSEC/DMSEC 2004*, pp. 65-72.

133. Langley, P. (1999) User Modeling in Adaptive Interfaces. *User Modeling 1999*, pp. 357-371.

134. Lanier, Jaron, (1996) My problems with agents, *Wired*, 1996.

135. Lau, T., Bergman, L., Castelli, V. and Oblinger, D. (2004) Sheepdog: Learning Procedures for Technical Support. *IUI 2004*, pp.109-116.

136. Lau, T., Domingos, P. and Weld. D. (2000) Version Space Algebra and Its Application to Programming by Demonstration. *ICML 2000*, pp. 527-534.

137. .Lederer, S., Hong, J.I., Dey, A.K., and Landay, J.A. (2004) Personal Privacy through Understanding and Action: Five Pitfalls for Designers. *Personal Ubiquitous Computing 8* (6): pp. 440-454.

138. Lee J.D. and See K.A. (2004) Trust in automation: designing for appropriate reliance. *Hum. Fact 42* (1): pp. 50–80.

139. Lesh, N., Rich, C. and Sidner, C. (1999) Using Plan Recognition in Human-Computer Collaboration. *User Modeling 1999*, pp. 23-32.

140. Li, C., Yan, N., Roy, S.B., Lisham, L. and Das, G. (2010) Facetedpedia: Dynamic Generation of Query-DependentFaceted Interfaces for Wikipedia. *WWW 2010*, pp. 651-660.

141. Licsar, A. and Sziranyi (2005) User Adaptive Hand Gesture Recognition System with Interactive Training. *Journal of Image and Vision Computing 23* (12): pp. 1102-1114.

142. Lieberman, H. (1993) Tinker: A Programming by Demonstration System for Beginning Programmers. In Allen Cypher (ed.) *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA, pp. 49-64.

143. Lieberman, H. (2000) *Your Wish is My Command: Giving Users the Power to Instruct their Software*. Morgan Kaufmann, 2000.

144. Lieberman (2009) User Interface Goals, AI Opportunities, *AI magazine 2009*.

145. Lieberman, H. and Espinosa, J. (2007) A Goal-Oriented Interface for Consumer Electronics Using Planning and Commonsense Reasoning. *Knowledge-Based Systems 20* (6): pp. 592-606.

146. Lim, B. and Dey, A. K. (2009) Assessing Demand for Intelligibility it Context-Aware Applications. *UbiComp 2009*, pp. 195-204.

147. Lim, B. and Dey, A. (2010) Toolkit to support intelligibility in context-aware applications. *UbiComp 2010*, pp. 13-22.

148. Lim, B., Dey, A.K. and Avrahami, D. (2009) Why and why not explanations improve the intelligibility of context-aware intelligent systems. *CHI 2009*, pp. 2119-2128.

149. Liu, G., Mok, A.K., and Yang, E.J. (1999) Composite Events for Network Event Correlation. *IEEE INM 1999*, pp. 247-260.

150. MacKay, D.J.C. (1992) Information-Based Objective Functions for Active Data Selection. *Neural Computation 4* (4): pp. 590-604.

151. MacLean, D., Hangal, S., Teh, S.K., Lam, M.S. and Heer, J. (2011) Groups Without Tears: Mining Social Topologies from Email. *IUI 2011*, pp. 83-92.

152. Maes, P. and Kozierok, R. (1993) Learning Interface Agents. *AAAI 1993*, pp. 459-465.

153. Manly, B.F.J. (1997) *Randomization, Bootstrap and Monte Carlo Methods in Biology, 2nd ed*. Chapman & Hall, NY, 1997.

154. Manning, C.D., Raghavan, P. and Schutze, H. (2008) *Introduction to Information Retrieval*, Cambridge University Press.

155. Mayhew, D.J. (1992) *Principles and Guidelines in Software User Interface Design*, Englewood Cliffs, NJ: Prentice Hall.

156. McCallum, A., Wang, X. and Corrada-Emmanuel, A. (2007) Topic and Role Discovery in Social Networks with Experiments on Enron and Academic Email. *Journal of Artificial Intelligence Research 30* (1): pp. 249–272.

157. McCarthy, K., Reilly, J., McGinty, L. and Smyth, B. (2005) Experiments in dynamic critiquing. *IUI 2005*, pp. 175-182.

158. McDaniel, R. and Myers, B.A. (1999) Getting More Out of Programming-by-Demonstration. *CHI 1999*, pp. 442-449.

159. McGinty, L. and Smyth, B. (2003) On the role of diversity in conversational recommender systems. In *ICCBR'03*, pp. 276-290.

160. McNee, S. Lam, C. Guetzlaff, J. Konstan, and J. Riedl. (2003) Confidence metrics and displays in recommender systems. *Interact'03*, pp. 176–183.

161. McSherry, D. (2004) Explanation in recommender systems. *Conference on Case-Based Reasoning 2004*, pp. 125-134.

162. Minka, T.P. and Picard, R.W. (1996) Interactive Learning with a Society of Models. *IEEE CVPR 1996*, pp. 447-452.

163. Mitchell, J. (2011) Making Photo Tagging Easier. *The Facebook Blog*, June 30, 2011.

164. Mitchell, J. and Shneiderman, B. (1989) Dynamic Versus Static Menus: An Exploratory Comparison. *SIGCHI Bulletin 20* (4): pp. 33–37.

165. Mitchell, T. (1997) *Machine Learning*. McGraw Hill.

166. Modugno, F., Corbett, A., and Myers, B. A. (1997) Graphical Representation of Programs in a Demonstrational Visual Shell—An Empirical Evaluation. *ACM TOCHI 4* (3): pp. 276–308.

167. Moggridge, B. (2007) *Designing Interactions*. MIT Press, Cambridge, MA.

168. Montaner, M., Lopez, B., de la Rosa, J.L. (2003) A Taxonomy for Recommender Agents on the Internet. *Artificial Intelligence Review 19,* pp. 285-330.

169. Mozer, M.C. (1998) The Neural Network House: An Environment that Adapts to its Inhabitants. *AAAI 1998 Spring Symposium on Intelligent Environments*, pp. 110-114.

170. Muir, B. (1994). Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems. *Ergonomics 37* (11): pp. 1905–1922.

171. Muir B.M. and Moray N. (1996) Trust in automation. Part II. *Ergonomics 39*, pp. 429-460.

172. Munoz, M., Gonzalez, V., Rodriguez, M. and Favela, J. (2003) Supporting Context-Aware Collaboration in a Hospital: An Ethnographic Informed Design. *Lecture Notes in Computer Science 2806/2003*, pp. 330-344.

173. Muramatsu, J., and Pratt, W. (2001) Transparent Search Queries: Investigating users' mental models of search engines. *SIGIR 2001*.

174. Myers, B. and Buxton, W. (1986) Creating Highly-Interactive and Graphical User Interfaces by Demonstration. *SIGGRAPH 1986*, pp. 249-258.

175. Nicolescu, M. and Mataric, M. (2001) Learning and interacting in human robot domains. *IEEE Transactions on Systems, Man, and Cybernetics 31* (5): pp. 419–430.

176. Norman, D. (1988) *The Design of Everyday Things*. New York: Basic Books.

177. Norman, D. A. (1994) How Might People Interact with Agents? *Communications of the ACM 37* (7): pp. 68–71.

178. Nwana, H. S. (1990) Intelligent Tutoring Systems: an Overview. *Artificial Intelligent Review 4*, pp. 251-277.

179. Olson, J.S., Grudin, J. and Horvitz, E. (2005) A Study of Preferences for Sharing and Privacy. *Ext. Abstracts CHI 2005*, pp. 1985-1988.

180. Ortutay, B. (2011) Facebook to Organize Friends in 'Smart Lists'. *USA Today,* Sept. 13, 2011. http://www.usatoday.com/tech/news/story/2011-09-13/facebook-smart-lists/50389154/1.

181. Pachal, P. (2011) Google Circles: The Dumbest Thing About Google+. *PC Mag,* June 29, 2011. http://www.pcmag.com/article2/0,2817,2387808,00.asp

182. Palen, L. and Dourish, P. (2003) Unpacking "Privacy" for aNetworked World. *CHI 2003*, pp. 129-136.

183. Parasuraman, R., Miller, C. (2004) Trust and etiquette in high-criticality automated systems. *Commun.ACM 47* (4), pp. 51–55.

184. Parker, A. (2011) Congressman, Sharp Voice on Twitter, Finds It Can Cut 2 Ways. *New York Times,* May, 30. 2011. http://www.nytimes.com/2011/05/31/nyregion/for-repanthony-weiner-twitter-has-double-edge.html.

185. Payne, T.R. and Edwards, P. (1997) Interface Agents that Learn: An Investigation of Learning Issues in a Mail Agent Interface. *Applied Artificial Intelligent 11*, Taylor & Francis, pp. 1-32.

186. Paynter, G.W. (2000) Automating Iterative Tasks with Programming by Demonstration. *Doctoral Thesis*, University of Waikato, February 2000.

187. Paynter, G., and Witten, I.H. (2004) Applying machine learning to programming by demonstration. *Journal of Experimental and Theoretical Artificial Intelligence 16* (3): pp. 161–188.

188. Pazzani, M. J. (2000) Representation of Electronic Mail Filtering Profiles: A User Study. *IUI 2000*, pp. 202-206.

189. Pennock, D. M. and Horvitz, E. (1999) Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach. *IJCAI 1999 Workshop on Machine Learning for Information Filtering*.

190. Pitkow, J., Schutze, H., Cass, T., Cooley, R., Turnbull, D., Edmonds, A., Adar, E. and Breuel, T. (2002) Personalized Search. *Communications of the ACM Magazine 45* (9): pp. 50-55.

191. Pogue, D. (2011) Google+ Improves on Facebook. *New York Times,* July 13, 2011.

http://www.nytimes.com/2011/07/14/technology/personaltech/google-gets-a-leg-up-on-facebook.html.

192. Pu, P. and Chen, Li. (2006) Trust building with explanation interfaces. *IUI 2006*, pp. 93-100

193. Raghavan, H. and Allan, J. (2007) An Interactive Algorithm for Asking and Incorporating Feature Feedback into Support Vector Machines. *SIGIR 2007*, pp. 79-86.

194. Rao, S., Hurlbutt, T., Nass, C., and JanakiRam, N. (2009) My Dating Site Thinks I'm a Loser: effects of personal photos and presentation intervals on perceptions of recommender systems. *CHI 2009*, pp. 221-224.

195. Rich, C. and Sidner, C. (1997) Segmented Interaction History in a Collaborative Interface Agent. *IUI 1997*, pp. 23-30.

196. Rich, C., Sidner, C., Lesh, N., Garland, A., Booth, S. and Chimani, M. (2005) DiamondHelp: A Graphical User Interface Framework for Human-Computer Collaboration. *IEEE Conference on Distributed Computing Systems*, pp. 514–519.

197. Ritter, A. and Basu, S. (2009) Learning to Generalize for Complex Selection Tasks. *IUI 2009*, pp. 167-176.

198. Rocchio, J. (1971) Relevance feedback information retrieval. In Salton, G. (Eds.), *The SMART Retrieval System: Experiments in Automatic Document Processing*. Englewood Cliffs, NJ: Prentice-Hall (1971), pp. 313-323.

199. .Rosenblum, D. (2007) What Anyone can Know: The Privacy Risks of Social Networking Sites. *IEEE Security & Privacy 5* (3): pp. 40-49.

200. Rooney, B. (2012) Big Data's Big Problem: Little Talent. *The Wall Street Journal*, April 29, 2012.

201. Ross, E. (2000) Intelligent User Interfaces: Survey and Research Directions. *Tech Report, University of Bristol*, Bristol.

202. Roth, M. Ben-David, A., Deutscher, D., Flysher, G., Horn, I., Leichtberg, A., Leiser, N., Matias, Y. and Merom, R. (2010) Suggesting Friends using the Implicit Social Graph. *KDD 2010*, pp. 233-242.

203. Ruvini, J. D. (2003) Adapting to the User's Internet Search Strategy. *User Modeling 2003*, pp. 55–64.

204. Sahami, M., Dumais, S., Heckerman, D. and Horvitz, E. (1998) A Bayesian Approach to Filtering Junk E-mail. *AAAI Workshop on Learning for Text Categorization, AAAI Tech Report WS-98-05.*

205. Salton, G. and Buckley, C. (1990) Improving Retrieval Performance by Relevance Feedback. *Journal of the American Society of Information Science 41*, pp. 288-297.

206. Schafer, B.J., Konstan, J.A., and Riedl, J. (2001) E-Commerce Recommendation Applications. *Journal of Data Mining and Knowledge Discovery 5* (1-2): pp. 115-153.

207. Schilit, B., Adams, N. and Want, R. (1994) Context-Aware Computing Applications. *IEEE Workshop on Mobile Computing Systems and Applications*, pp. 85-90.

208. Secord, A., Winnemoeller, H., Li, W. and Dontcheva, M. (2010) Creating Collections with Automatic Suggestions and Example-Based Refinement. *UIST 2010*, pp. 249-258.

209. Segal, R. B. and Kephart, J. O. (1999) MailCat: An intelligent Assistant for Organizing E-mail. *AGENTS 1999*, pp. 276-282.

210. Sheehan, K. (2002) Towards a Typology of Internet Users and Online Privacy Concerns. *The Information Society 18*, pp. 21-23.

211. Shen, X., Tan, B. and Zhai, C. (2005) Implicit User Modeling for Personalized Search. *CIKM 2005*, pp. 824–831.

212. Shilman, M., Tan, D. S. and Simard, P. (2006) CueTip: A Mixed-Initiative Interface for Correcting Handwriting Errors. *UIST 2006*.

213. Shneiderman, B. and Maes, P. (1997) Direct manipulation vs. interface agents. *Interactions 4* (6): pp. 42–61.

214. Siegler, M. (2010) Zuckerberg: "Guess What? Nobody Wants to Make Lists." *TechCruch.com,* Aug. 26, 2010. http://techcrunch.com/2010/08/26/facebook-friend-lists/.

215. Sinha, R. and Swearingen, K. (2002) The role of transparency in recommender Systems. *CHI 2002 Extended Abstracts*

216. Slee, M. (2007) Friend Lists. *Facebook Blog,* Dec. 19, 2007. http://www.facebook.com/blog.php?post=7831767130.

217. Smetters, D.K. and Good, N. (2009) How Users Use Access Control. *SOUPS 2009*.

218. Smyth, B. and Balfe, E. (2006) Anonymous Personalization in Collaborative Web Search. *Information Retrieval 9* (2): pp. 165-190.

219. Spaulding, A., Blythe, J., Haines, W. and Gervasio, M. (2009) From Geek to Sleek: Integrating Task Learning Tools to Support End Users in Real-World Applications. *IUI 2009*, pp. 389-394.

220. Speretta, M. and Gauch, S. (2005) Personalized Search Based on User Search Histories. *IEEE/WIC/ACM Conference on Web Intelligence 2005*, pp. 622–628.

221. Spring, N., Mahajan, R., Wetherall, D., and Anderson, T. (2002) Measuring ISP Topologies with Rocketfuel. *SIGCOMM 2002*, pp. 133-145.

222. Stamou, S. and Ntoulas, A. (2009) Search Personalization Through Query and Page Topical Analysis. *User Modeling and User-Adapted Interaction 19* (1-2): pp. 5-33.

223. Steinder, M. and Sethi, A.S. (2004) A Survey of Fault Localization Techniques in Computer Networks. *Science of Computer Programming 53,* pp. 165-194.

224. Stumpf S, Rajaram V, Li L, Burnett M, Dietterich T, Sullivan E, Drummond R, Herlocker J.(2007) Toward Harnessing User Feedback For Machine Learning. *IUI 2007*.

225. Stumpf, S., Sullivan, E., Fitzhenry, E., Oberst, I., Wong, W., Burnett, M. (2008) Integrating rich user feedback into intelligent user interfaces. *IUI 2008*, pp. 50-59.

226. Swearingen and R. Sinha. (2002) Interaction design for recommender systems. *DIS2002*.

227. Teevan, J., Dumais, S. and Horvitz, E. (2005) Personalizing Search Via Automated Analysis of Interests and Activities. *SIGIR 2005,* pp. 449-456.

228. Tenenbaum, J.B., de Silva, V. and Langford, J.C. (2000) A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science 290* (5500): pp. 2319-2323.

229. Tintarev, N. and Masthoff, J. (2007) Effective explanations of recommendations: User-centered design. *RecSys 2007*, pp. 153–156.

230. Tintarev, N. and Masthoff, J.(2007) Survey of explanations in recommender systems. *WPRSIUI 2007*.

231. Tong, S. and Chang, E. (2001) Support Vector Machine Active Learning for Image Retrieval. *Multimedia 2001*, pp. 107-118.

232. Toomim, M., Zhang, X., Fogarty, J. and Landay, J. (2008) Access Control by Testing for Shared Knowledge. *CHI 2008*, pp. 193-196.

233. Tullio, J., Dey, A., Chalecki, J. and Fogarty, J. (2007) How it works: a field study of non-technical users interacting with an intelligent system. *CHI 2007*, pp. 31-40.

234. van Zwol, R., Sigurbjörnsson, B., Adapala, R., Pueyo, L.G., Katiyar, A., Kurapati, K., Muralidharan, M., Muthu, S., Murdock, V., Ng, P., Ramani, A., Sahai, A., Sathish, S.T., Vasudev, H. and Vuyyuru. U. (2010) Faceted exploration of image search results. *WWW 2010,* pp. 961-970.

235. Verpoorten, K., Luyten, K. and Coninx, K. (2007) Mixed Initiative Ambient Environments: A Self-Learning System to Support User Tasks in Interactive Environments. *Workshop on Context Awareness for Proactive Systems (CAPS 2007)*.

236. Victor, P., Cornelis, C., De Cock, M., da Silva, P.P. (In Press) Gradual Trust and Distrust in Recommender Systems. *Fuzzy Sets and Systems*.

237. Vig, J., Sen, S. and Riedl, J. (2009) Tagsplanations: explaining recommendations using tags. *IUI 2009*, pp. 47-56.

238. Waern, A. (2004) User Involvement in Automatic Filtering: an Experimental Study. *UMUAI 14* (2-3): pp. 201-237.

239. Want, R., Hopper, A., Falcao, V. and Gibbons, J. (1992) The Active Badge Location System. *ACM TOIS 10* (1): pp. 91-102.

240. Ware, M., Frank, E., Holmes, G., Hall, M., Witten, I. H. (2001) Interactive machine learning: letting users build classifiers. *IJHCS 55*, pp. 281-292.

241. Webb, G., Pazzani, M. and Billsus, D (2001) Machine Learning for User Modeling. *UMUAI 11*, pp. 19-20.

242. Weiss, R. and Zgorski, L. (2012) Obama Administration Unveils “Big Data” Initiative: Announces $200 Million in New R&D Investments. *White House Press Release*, March 29, 2012.

243. Whalen, T., Smetters, D., and Churchill, E.F. (2006) User Experiences with Sharing and Access Control. *CHI 2006 Extended Abstracts*.

244. White, R. (2005) Implicit Feedback for Interactive Information Retrieval. *Doctoral Dissertation*, University of Glasgow, Glasgow.

245. White, R., Ruthven, I. and Jose, J.M. (2002) The User of Implicit Evidence for Relevance Feedback in Web Retrieval. *Advances in Information Retrieval*, pp. 93-109.

246. Winograd, T. (1996) *Bringing Design to Software*. ACM Press, New York.

247. Witten, I.H. and Frank, E. (2005) *Data Mining: Practical Machine Learning Tools and Techniques (2nd Edition)*, Morgan Kaufmann, San Francisco, CA, USA.

248. Wobbrock, J.O., Findlater, L., Gergle, D. and Higgins, J.J. (2011) The Aligned Rank Transform for Nonparametric Factorial Analyses using only ANOVA Procedures. *CHI 2011*, pp. 143-146.

249. Wolfman, S., Lau, T., Domingos, P. and Weld, D.S. (2001) Mixed Initiative Interfaces for Learning Tasks: SMARTedit Talks Back. *IUI 2001*, pp. 167-174.

250. World Economic Forum Report (2012) Big Data, Big Impact: New Possibilities for International Development. 2012. http://www3.weforum.org/docs/WEF_TC_MFS_BigDataBigImpact_Briefing_2012.pdf

251. Yan, T.W. and Garcia-Molina, H. (1999) The SIFT Information Dissemination System. *ACM Transactions on Database Systems 24* (4): pp. 529-565.

252. Yemini, S., Kliger, S., Mozes, E., Yemini, Y., and Ohsie, D. (1996) High Speed and Robust Event Correlation. *IEEE Communications Magazine 34* (5): pp. 82-90.

253. Zammit-Mangion, A., Dewar, M., Kadirkamanathan, V. and Sanguinetti, G. (2012) Point Process Modeling of the Afghan War Diary. *PNAS 2012*.

254. Zaslow, J. (2002) "If TiVo Thinks You Are Gay, Here's How To Set It Straight - Amazon.com Knows You, Too, Based on What You Buy; Why All the Cartoons?" *The Wall Street Journal*, November 26, 2002.

255. Zeng, H-J., He, Q-C., Chen, Z., Ma, W-Y. and Ma, J. (2004) Learning to Cluster Web Search Results. *SIGIR 2004*, pp. 210-217.

256. Zhang, Y., Callan, J. and Minka, T. (2002) Novelty and Redundancy Detection in Adaptive Filtering. *SIGIR 2002*, pp. 81-88.

257. Zhu, X. (2005) Semi-supervised learning literature survey. *Technical Report 1530*, Department of Computer Sciences, University of Wisconsin, Madison.