City Research Online

City St George's, University of London

Citation: Kulesza, T., Burnett, M., Wong, W-K. & Stumpf, S. (2015). Principles of Explanatory Debugging to personalize interactive machine learning. In: Brdiczka, O. & Chau, P (Eds.), Proceedings of the 20th International Conference on Intelligent User Interfaces. (pp. 126-137). New York, USA: ACM. ISBN 9781450333061 doi: 10.1145/2678025.2701399

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/13819/

Link to published version: https://doi.org/10.1145/2678025.2701399

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to City Research Online policy.

City Research Online: http://openaccess.city.ac.uk/ publications@citystgeorges.ac.uk

Principles of Explanatory Debugging to Personalize Interactive Machine Learning

ABSTRACT

How can end users efficiently influence the predictions that machine learning systems make on their behalf? This paper presents Explanatory Debugging, an approach in which the system explains to users how it made each of its predictions, and the user then explains any necessary corrections back to the learning system. We present the principles underlying this approach and a prototype instantiating it. An empirical evaluation shows that Explanatory Debugging increased participants' understanding of the learning system by 52% and allowed participants to correct its mistakes up to twice as efficiently as participants using a traditional learning system.

Author Keywords

Interactive machine learning; end user programming.

H.5.2. [**Information interfaces and presentation (e.g. HCI)**] User interfaces

INTRODUCTION

In many machine learning applications, the user has a concept in mind—songs he wants to hear, junk mail she doesn't want to see, anomalous transactions he wants to detect—that the machine learning system is trying to identify. An increasing body of research has begun to explore what users can do when their machine learning systems misidentify their concepts [e.g., 1, 2, 17, 24, 35, 42]. Much of this work shares a common challenge: how can end users effectively and efficiently personalize the predictions or recommendations these learning systems make on their behalf?

For example, engineer Alice has email folders for her project group at work, an office sports pool, and individual folders for colleagues she frequently collaborates with; she wants her email program to automatically categorize new email into the appropriate folder. This requires fine-grained control—perhaps she wants email from her colleague Bob to go to the "sports" folder if it mentions something about hockey, the "project" folder if it references something related to their current project, or the "Bob" folder if neither of those conditions are met. However, Alice does not have enough email from Bob to accurately train a learning system to recognize these three concepts, and by the time she acquires it, the hockey season may be over or her group's project may change.

Our proposed solution is *Explanatory Debugging*, which is an explanation-centric approach to help end users effectively and efficiently personalize machine learning systems. We say "debugging" because personalizing a machine learning system is an end-user debugging problem: the user is trying to exert fine-grained control over the system's learned behavior.

In Explanatory Debugging, the system explains the reasons for its predictions to its end user, who in turn explains corrections back to the system. We hypothesize that this cycle of explanations will help users build useful *mental models*—internal representations that allow people to predict how a system will behave [16]—and thus allow them to personalize their learning system better and faster than traditional black-box systems.

In the earlier example, such an approach could show Alice all the words the system uses to identify email about her group's project, the sports pool, or any other concept. She could then add additional words she knows to be relevant, or remove words that she knows are irrelevant. She would not need to worry that her corrections might confuse the system (e.g., telling it that a message from Bob about hockey belongs in the sports folder and hoping it will identify that the topic is the reason, not the sender), nor would she need to spend weeks or months acquiring new training instances.

In this paper we present the Explanatory Debugging approach and instantiate it in EluciDebug, the first interactive system designed to help end users build useful mental models of a machine learning system while simultaneously allowing them to explain corrections back to the system. We evaluate Explanatory Debugging against a traditional black-box learning system to determine whether Explanatory Debugging helped end users build better mental models, and if so, whether these mental models helped users more efficiently personalize their learning system's behavior.

RELATED WORK

The study of how end users interactively control machine learning systems is gaining increased attention. Fails et al. first popularized the phrase *interactive machine learning* in a paper describing how an iterative train-feedback-correct cycle allowed users to quickly correct the mistakes made by an

image segmentation system [11]. Since then, researchers have explored using this cycle of quick interactions to train instance-based classifiers [5, 13], enable better model selection by end users [2, 12, 37], elicit labels for the most important instances (e.g., active learning) [7, 34], and to improve reinforcement learning for automated agents [20]. These use cases largely treat the machine learning system as a "black box"—users can try to personalize the system by providing it with different inputs (e.g., labeled instances), but are unable to see *why* different inputs may cause the system's outputs to change.

An alternative to the "black box" model is a "white box" model [15], in which the machine learning system is made more transparent to the user. Increased transparency has been associated with many benefits, including increased user acceptance of (or satisfaction with) recommendations and predictions [4, 8, 39], improved training of intelligent agents [38], and increased trust in the system's predictions [10]. In this paper, however, we primarily care about transparency because it has been shown to help users understand how a learning system operates [23, 28].

Recent work has suggested that as end users better understand how a machine learning system operates, they are better able to personalize it [12, 21]. Further, research has shown that in the absence of transparency, it is difficult to build accurate mental models of learning systems [21, 40]. People will still build mental models of the system, but such flawed mental models often lead to misinformed user behavior [18, 30].

Some researchers have sought to expand the types of inputs interactive machine learning systems can process, and this can also increase system transparency. For example, user feedback may take the form of model constraints [17], critiques to the model's search space [42], or adjustments to the model's weights of evidence [24]. In addition to supporting novel user feedback mechanisms, these approaches increase transparency by enabling users to view such facets as the existing model constraints [17], the model's feature set [42], or the weights of evidence responsible for each prediction [24].

All of these works have increased our understanding of how human factors impact machine learning, but none have evaluated the ability of an interactive machine learning system to (1) help users build useful mental models of how it operates such that (2) they can efficiently personalize the system. Kulesza et al. found a link between mental models and personalization ability, but relied on human instructors to help participants build useful mental models [21]. Other work has studied the impact of machine-generated explanations on users' mental models, but did not explore whether this helped participants' personalize their learning systems [8, 23, 28]. In this paper we rely on machine-generated explanations to help users build good mental models and then evaluate their resulting ability to personalize a learning system.

EXPLANATORY DEBUGGING

In 2010, Kulesza et al. proposed the idea of Explanatory Debugging—a two-way exchange of explanations between an end user and a machine learning system [22]—but did not provide principles underlying the approach. In this section we build upon that proposal by describing principles to ground and support replication of Explanatory Debugging.

Explainability

Our first principle for Explanatory Debugging is *Explainability*: accurately explain the learning system's reasons for each prediction to the end user. This principle builds upon research showing that end users who received explanations about a learning system's rules of behavior were able to better personalize classifiers and recommenders [4, 17, 22]. Without explanations, however, users struggle to build accurate mental models of such systems [21, 28, 40]. This suggests that explanations are a necessary condition to help end users learn how a machine learning system operates. To help users build useful mental models of the learning system, these explanations should observe the following principles:

Principle 1.1: Be Sound

Explanations should not be simplified by explaining the model as if it were less complex than it actually is. In [23] this is referred to as *soundness*: "the extent to which each component of an explanation's content is truthful in describing the underlying system." In [21] the authors found a linear correlation between the quality of a user's mental model and their ability to control the learning system as desired, suggesting that the better someone understands the underlying system, the better they will be able to control it. Further, [23] details the impact of explanation fidelity on mental model development, finding that users did not trust—and thus, were less likely to attend to—the least sound explanations. Because reducing soundness reduces both the potential utility of the explanation and the likelihood that users will invest attention toward it, Explanatory Debugging entails designing explanations that are as sound as practically possible.

One method for evaluating explanation soundness is to compare the explanation with the learning system's mathematical model. How accurately are each of the model's terms explained? If those terms are derived from more complex terms, is the user able to "drill down" to understand those additional terms? The more these explanations reflect the underlying model, the more sound the explanation is.

Principle 1.2: Be Complete

Kulesza et al. describe *completeness* as "the extent to which *all* of the underlying system is described by the explanation" [23], so a complete explanation does not omit important information about the model. In that study, end users built the best mental models when they had access to the most complete explanations, which informed them of all the information the learning system had at its disposal and how it used that information [23]. Also pertinent is work showing that users often struggle to understand how different parts of the system interact with each other [24]. Complete explanations that reveal how different parts of the system are interconnected may help users overcome this barrier.

One method for evaluating completeness is via Lim and Dey's *intelligibility types* [27], with more complete explanations including more of these intelligibility types.

Principle 1.3: But Don't Overwhelm

Balanced against the soundness and completeness principles is the need to remain comprehensible and to engage user attention. Findings from [23] suggest that one way to engage user attention is to frame explanations concretely, such as referencing the predicted item and any evidence the learning system employed in its prediction. In some circumstances, selecting a more comprehensible machine learning model may also be appropriate. For example, a neural network can be explained as if it were a decision tree [9], but this reduces soundness because a different model is explained. Similarly, a model with 10,000 features can be explained as if it only used the 10 most discriminative features for each prediction, but this reduces completeness by omitting information that the model uses. Alternative approaches that embody the Explanatory Debugging principles include selecting a machine learning model that can be explained with little abstraction [e.g., 25, 35, 36] or using feature selection techniques [44] in high-dimensionality domains to prevent users from struggling to identify which features to adjust (as happened in [24]).

Correctability

Our second top-level principle for Explanatory Debugging is *Correctability*: allow users to explain corrections back to the learning system. To enable an iterative cycle of explanations between the system and the user, in Explanatory Debugging the machine-to-user explanation should also serve as the user-to-machine explanation. Research suggests that to elicit corrections from users, this feedback mechanism should embody the following principles:

Principle 2.1: Be Actionable

Both theory [3] and prior empirical findings [6, 21, 23] suggest end users will ignore explanations when the benefit of attending to them is unclear. By making the explanation actionable, we hope to lower the perceived cost of attending to it by obviating the need to transfer knowledge from one part of the user interface (the explanation) to another (the feedback mechanism). Actionable explanations also fulfill three aspects of Minimalist Instruction [41]: (1) people are learning while performing real work; (2) the explanatory material is tightly coupled to the system's current state; and (3) people can leverage their existing knowledge by adjusting the explanation to match their own mental reasoning.

Principle 2.2: Be Reversible

A risk in enabling users to provide feedback to a machine learning system is that they may actually make its predictions worse [e.g., 22, 35]. Being able to easily reverse a harmful action can help mitigate that risk. It may also encourage self-directed tinkering, which can facilitate learning [33]. When combined with Principle 2.4, reversibility also fulfills a fourth aspect of Minimalist Instruction [41]: help people identify and recover from errors.

Principle 2.3: Always Honor User Feedback

As Yang and Newman found when studying users of learning thermostats [43], a system that appears to disregard user feedback deters users from continuing to provide feedback. However, methods for honoring user feedback are not always straightforward. Handling user feedback over time (e.g., what if new instance-based feedback contradicts old instance-based feedback?) and balancing different types of feedback (e.g., instance-based feedback versus feature-based feedback) requires careful consideration of how the user's feedback will be integrated into the learning system.

Principle 2.4: Incremental Changes Matter

In [23], participants claimed they would attend to explanations *only if* doing so would enable them to more successfully control the learning system's predictions. Thus, continued user interaction may depend upon users being able to see incremental changes to the learning system's reasoning after each interaction (i.e., overcoming the *gulf of evaluation* that exists between a user's mental model and a system's actual state [31]). Additionally, our thesis is that users will develop better mental models iteratively, requiring many interactions with the learning system. These interactions may not always result in large, obvious changes, so being able to communicate the small, incremental changes a user's feedback has upon a learning system seems critical to our approach's feasibility.

ELUCIDEBUG: EXPLANATORY DEBUGGING IN ACTION

We instantiated Explanatory Debugging in a text classification prototype we call EluciDebug (Figure 1). We chose text classification because (1) many real-world systems require it (e.g., SPAM filtering, news recommendation, serving relevant ads, search result ranking, etc.) and (2) it can be evaluated with documents about common topics (e.g., popular sports), allowing a large population of participants for our evaluation. We designed EluciDebug to look like an email program with multiple folders, each representing a particular topic. The prototype's machine learning component attempts to automatically classify new messages into the appropriate folder.

While the principles of Explanatory Debugging primarily deal with the user interface, two principles place constraints on the machine learning model: (1) it must be able to *honor user feedback* in real-time, and (2) it must be explainable with enough soundness and completeness to allow users to build useful mental models of how it operates *without overwhelming* them. Our EluciDebug prototype uses a multinomial naive Bayes model (MNB) [19] with feature selection [44] to meet these constraints. Evaluating the suitability of—or changes necessary to—other models remains an open question.

The Multinomial Naive Bayes Classifier: A Brief Review

Before describing how we integrated MNB with Explanatory Debugging, we first summarize how MNB operates. An MNB classifier computes the probability that a given input (e.g., the document being classified) has of belonging to each output (e.g., the possible labels). The output with the highest probability "wins" and becomes the predicted label for the input. For example, if MNB calculates that a document has a 70% probability of being junk mail and a 30% probability of not being junk mail, the document will be labeled as junk mail. The equations for computing probability, as defined in [19], are shown below. We use c to represent an individual class

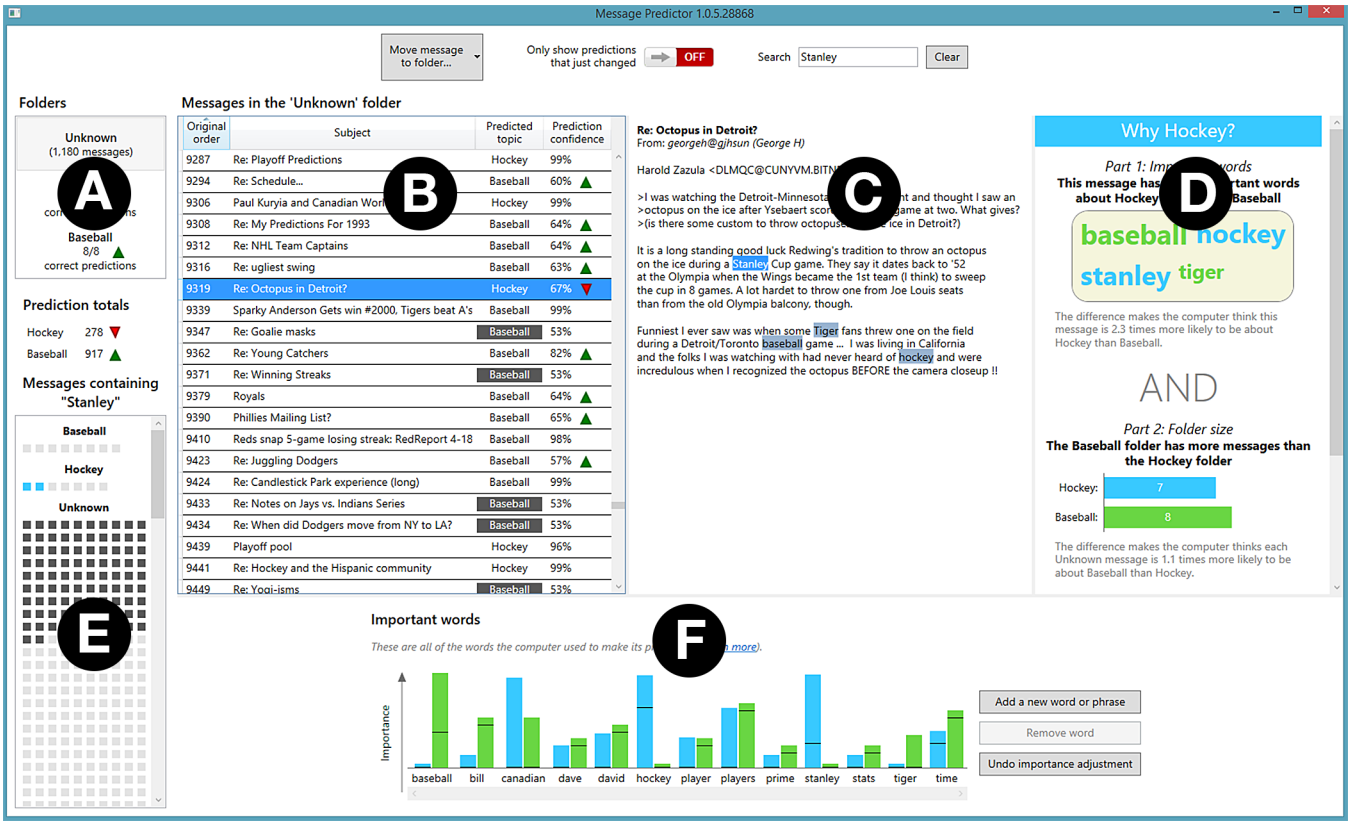

Figure 1. The EluciDebug prototype. (A) List of folders. (B) List of messages in the selected folder. (C) The selected message. (D) Explanation of the selected message's predicted folder. (E) Overview of which messages contain the selected word. (F) Complete list of words the learning system uses to make predictions.

in the collection of potential output classes C, d_i to represent an individual document to classify, and assume that the only features the classifier uses are individual words in the set of known documents:

$$\Pr(c|d_i) = \frac{\Pr(c)\Pr(d_i|c)}{\Pr(d_i)} \quad (1)$$

The term $\Pr(c)$ represents the probability that any given document belongs to class c and can be estimated by dividing the number of documents in c by the total number of documents in the training set. The term $\Pr(d_i|c)$ represents the probability of document d_i given class c and can be estimated as:

$$\Pr(d_i|c) = \prod_n \Pr(w_n|c)^{f_{ni}} \quad (2)$$

The term f_{ni} is the number of instances of word n in document d_i and the term $\Pr(w_n|c)$ is the probability of word n given class c, estimated with the equation

$$\Pr(w_n|c) = \frac{p_{nc} + F_{nc}}{\sum_{x=1}^{N} p_{xc} + \sum_{x=1}^{N} F_{xc}} \quad (3)$$

where F_{nc} is the number of instances of word n in all of the training documents for class c, N is the number of unique words in the training documents for all classes, and p_{nc} is a smoothing term (usually 1) to prevent the equation from yielding 0 if no documents from class c contain word w_n.

The Explanatory Debugging Principles in EluciDebug

Being Sound

Soundness means that everything an explanation says is true. Our explanations aim to be sound by accurately disclosing all of the features the classifier used to make its prediction, as well as how each feature contributed to the prediction. EluciDebug's *Why* explanation is responsible for communicating much of this information to users (Figure 2).

Soundly explaining the MNB classifier requires explaining the $\Pr(c)$ and $\Pr(d_i|c)$ terms from Equation 1, as these are the only terms that impact the model's predictions[1]. Because the $\Pr(d_i|c)$ term expands into Equation 2, we also need to explain how word probabilities for each class factor into the prediction. We can further increase soundness by explaining how these probabilities are calculated (i.e., by explaining Equation 3).

In EluciDebug we explain the $\Pr(d_i|c)$ term by using a word cloud to visualize the difference between each feature's probability for the two output classifications (Equation 3) as the following ratio: $\frac{\Pr(w_n|c_1)^{f_{ni}}}{\Pr(w_n|c_2)^{f_{ni}}}$, where c_1 is the class with the larger probability for feature w_n. We use the result to compute

[1]The $\Pr(d_i)$ term in the denominator of Equation 1 is only used to normalize the result to fall within the range 0–1; it does not impact the model's prediction.

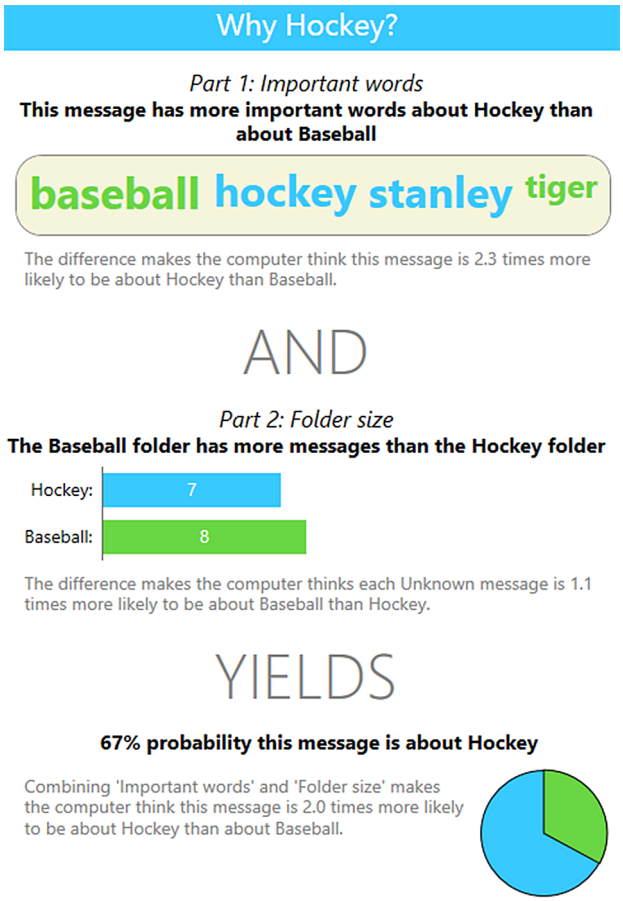

Figure 2. The *Why* explanation tells users how features and folder size were used to predict each message's topic. This figure is a close-up of Figure 1 part D.

the feature's font size, while its font color is based on the class with the larger probability. For example, in Figure 2 the word *stanley* is larger than *tiger* because its ratio of word probability is correspondingly larger, and it is blue because its probability of occurring in the *hockey* class is larger than its probability of occurring in the *baseball* class. Hovering over a word in this cloud shows a tooltip that explains the word's size was determined by a combination of (1) how many times it appeared in each class, and (2) any adjustments the user made to the word's importance (Equation 3).

The second component of a sound explanation of the MNB classifier is the $\Pr(c)$ term from Equation 1. We explain this term via a bar graph visualization of the number of items in each class (Figure 2, middle).

The top-to-bottom design of this entire *Why* explanation, along with the text that describes *Part 1* and *Part 2*, is intended to teach the user that both word presence (the $\Pr(d_i|c)$ term) and folder size (the $\Pr(c)$ term) play a role in each of the classifier's predictions. The result, shown at the bottom of Figure 2, explains how these two parts are combined to determine the classifier's certainty in its prediction.

Being Complete

Completeness means telling the *whole* truth. For the MNB classifier, this means not only explaining each of the terms from Equation 1, but also all of the information the classifier *could* use, where it came from, and how likely it is that each prediction is correct. To help ensure completeness, we turn to Lim and Dey's schema of intelligibility types. The results of [23] suggest that a complete explanation should include Lim and Dey's *why*, *inputs*, *model*, and *certainty* types. Lim's work suggests the usefulness of the *what if* type in scenarios where the user is attempting to change the behavior of a classifier [26], so we included this intelligibility type as well.

We thus designed our EluciDebug explanations to detail all of the information the classifier could potentially use when making predictions. The *Why* explanation shown in Figure 2 tells users that both feature presence and folder size played a role in each prediction (the numerator of Equation 1). The *Important words* explanations (Figure 4) goes even further, telling the user *all* of the features the classifier knows about and may use in its predictions. Because it tells the user about the sources of information available to the classifier, this is also an instantiation of Lim and Dey's *inputs* intelligibility type. To make it clear to users that these features can occur in all parts of the document—message body, subject line, and sender—EluciDebug highlights features in the context of each message (Figure 1, part C).

In [23] the authors found that Lim and Dey's *model* intelligibility type was associated with better mental models, but this intelligibility type was rarely attended to by most participants. To solve this dilemma, we incorporated the *model* content into our *Why* explanation—it explains all of the evidence the classifiers used, but it also explains where that evidence came from (e.g., words or folder size) and how it was combined to arrive at a final prediction. This approach has the added advantage of making the potentially abstract *model* intelligibility type very concrete; it is now tailored to each specific prediction.

A further aspect of completeness is evident in the *Feature overview* explanation (Figure 1, part E). This explanation shows users how many messages contain a given feature and is intended to help users identify when a feature they think the computer should pay attention to may not be very useful. The explanation updates in real-time as the user types in a potential feature; users do not need to add the feature to view its potential impact on classifications, making this an instance of the *what if* intelligibility type.

Finally, we also included the *certainty* intelligibility type. This is instantiated via the *Prediction confidence* column (Figure 1, part B), which reveals the classifier's confidence in each of its predictions to the user.

Not Overwhelming

To avoid overwhelming users, EluciDebug limits the initial set of features available to the classifier using information gain [44]. Because Principle 1.2 states that explanations should be as complete as possible, users should be able to see all of the classifier's features. Given this constraint, we decided

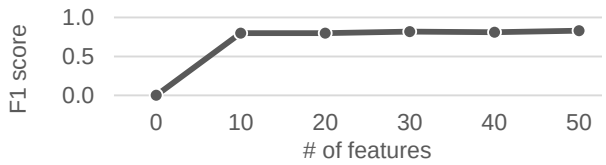

Figure 3. Selecting the 10 highest information gain features resulted in similar classifier performance as larger feature sets.

50 would be the upper limit on feature set size. Offline tests, however, revealed that MNB's F_1 score did not improve while the feature set size increased from 10 to 50 (Figure 3), so we decided our classifier would automatically select only the 10 features with the highest information gain (until the user specifies otherwise by adding or removing features).

Being Actionable

The *Important words* explanation (Figure 4) is the most actionable of EluciDebug's explanations. Users can add words to—and remove words from—this explanation, which in turn will add those words to (or remove them from) the machine learning model's feature set. Users are also able to adjust the importance of each word in the explanation by dragging the word's bar higher (to make it more important) or lower (to make it less important), which then alters the corresponding feature's weight in the learning model.

The *Why* explanation (Figure 2) is a likely candidate for actionability, but we have not yet made it actionable in EluciDebug. As this explanation includes only features extant in the selected message, it cannot replace the *Important words* explanation because doing so would interfere with our explanations' completeness. It could, however, complement the *Important words* explanation by allowing users to directly adjust the importance of features responsible for the given prediction. For example, users could drag out from the center of a word to increase its importance, or drag in toward the center of the word to decrease its importance. Whether such additional actionability would help users, however, remains an open question.

Being Reversible

EluciDebug includes an *undo* button for reversing changes to the *Important words* explanation. There is no limit on how many actions can be un-done because we want users to interact with the system without fear they will harm its predictions; regardless of how much they adjust its reasoning, they can always return to any prior state.

Honoring User Feedback

EluciDebug allows users to provide two types of feedback: traditional *instanced-based feedback*, where the user applies a label[2] to an entire item, and *feature-based feedback*, where the user tells the classifier an item should be labeled in a certain manner because of specific features it contains or feature values it matches. EluciDebug honors instance-based feedback in a straightforward manner: once the user labels an item, the classifier will use it as part of its training set, with no distinction between older versus more recent feedback. Honoring feature-based feedback, however, is more complicated.

[2]A *label* is a potential output of the learning system, such as *junk mail* or *normal mail*.

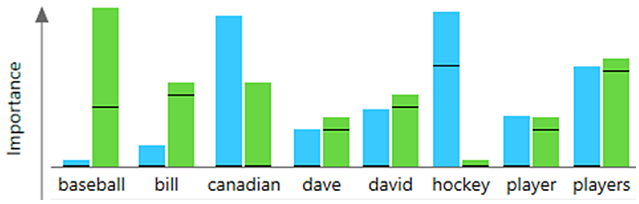

Figure 4. The *Important words* explanation tells users all of the features the classifier is aware of, and also lets users add, remove, and adjust these features. Each topic is color-coded (here, blue for *hockey* and green for *baseball*) with the difference in bar heights reflecting the difference in the word's probability with respect to each topic (e.g., the word *canadian* is roughly twice as likely to appear in a document about *hockey* as one about *baseball*, while the word *player* is about equally likely to appear in either topic). This figure is an excerpt from Figure 1 part F.

The smoothing term p_{nc} from Equation 3 acts as a Bayesian prior, effectively adding some number of virtual occurrences (again, traditionally 1) to the number of actual occurrences of each word in the training data, and we can leverage it to integrate feature-based feedback. By allowing the user to set the value of p_{nc}, we are letting the user increase the number of virtual occurrences of word n in class c. The result is a classifier that considers word n to be stronger evidence in favor of class c than it had before the user's feedback.

Using the smoothing term as a feature-based feedback mechanism, however, has a drawback: F_{nc} may increase as the training set size increases, causing the value of p_{nc} to become a smaller component of Equation 3. Thus, a user's feature-based feedback could account for less and less of the classifier's reasoning as their instance-based feedback increased.

To prevent a user's feature-based feedback from degrading in importance over time, we developed a visual feedback mechanism (Figure 4) that allows users to specify how important their feedback should be relative to the model's internal word probabilities (the F terms in Equation 3). The black lines on the blue and green bars in Figure 4 show the model-computed probabilities for each feature, which serve as a starting point for feature-based feedback. Users can tell the system that the probability of seeing word w_n in class c should be increased by clicking and dragging its bar higher, which will translate to an increased value for p_{nc}. If the user later provides additional instance-based feedback (thus causing F_{nc} to change), p_{nc} will be automatically recalculated such that the ratios of $\sum_{x=1}^{N} p_{xc}$ to $\sum_{x=1}^{N} F_{xc}$ and p_{nc} to F_{nc} remain constant.

Revealing Incremental Changes

There are many components of EluciDebug that may change after each user action, so to avoid confusing users with several different explanation paradigms, we designed a method that consistently reveals changes in terms of *increases* and *decreases*. Increases of any numeric value are identified by green

"up" arrows, while decreasing numeric values are identified by red "down" arrows. Examples of each are shown in Figure 1, part B. Hovering over either of these arrow icons yields a tooltip detailing what just changed and how much it changed by, e.g., "Confidence increased by 9%". These indicators reveal changes in the number of messages correctly classified in each folder, the total number of messages the machine learning model currently classified into each folder, and the confidence of each prediction.

In addition to numeric change indicators, we also needed an ordinal change indicator to highlight when a message's prediction flipped from one topic to the other. We used a grey background for these recently-changed predictions (see part B of Figure 1) and included a tooltip explaining that the user's last action resulted in the message's predicted topic changing.

EVALUATION

We evaluated Explanatory Debugging, as instantiated in EluciDebug, to investigate the following research questions:

RQ1: Does Explanatory Debugging help users personalize a classifier *more efficiently* than instance labeling?

RQ2: Does Explanatory Debugging help users personalize a classifier *more accurately* than instance labeling?

RQ3: Does Explanatory Debugging help users build *better mental models* than a traditional black-box approach?

Experiment Design

We used a between-subject, single-factor experimental setup to evaluate Explanatory Debugging. The factor we varied was experiment condition: one condition (*control*) used a variation of EluciDebug with all of its explanation and feature-based feedback capabilities removed (Figure 5), while the second condition (*treatment*) used the EluciDebug prototype described earlier. In both conditions EluciDebug was setup as a binary classifier that attempted to predict whether each message belonged to one of two topics.

To provide messages for EluciDebug, we selected two concepts from the 20 Newsgroups dataset[3]: *hockey* and *baseball* (the rec.sport.hockey and rec.sport.baseball newgroups, respectively). We used two subgroups of a larger group of related concepts (rec.sport) to ensure overlap in the terminology of each concept (e.g., "player" and "team" may be equally representative of *baseball* or *hockey*). These shared terms help make the classification task more challenging.

To simulate a situation where personalization would be required because sufficient training data does not yet exist, we severely limited the size of the machine learning training set for this experiment. At the start of the experiment this training set consisted of 5 messages in the *Hockey* folder and 5 messages in the *Baseball* folder, with 1,185 unlabeled messages in the *Unknown* folder. The small training set (10 messages) allowed us to evaluate a situation with limited training data, and the large amount of *potential* training data

[3]http://qwone.com/~jason/20Newsgroups/

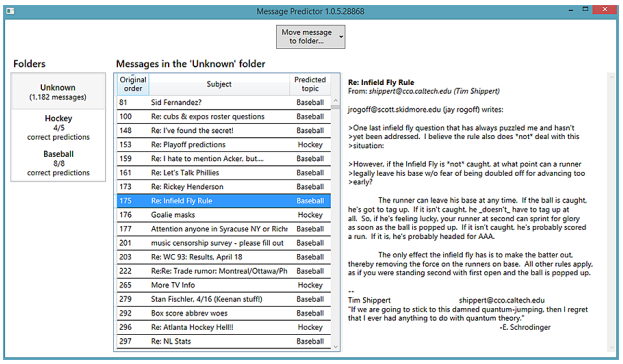

Figure 5. Control participants used this variant of EluciDebug, which lacks explanations and feature-based feedback.

(1,185 messages) allowed us to contrast Explanatory Debugging against black-box instance labeling in a situation where instance labeling could be expected to succeed.

Participants and Procedure

We recruited 77 participants (27 females, 50 males) from the local community and university. To ensure that participants would have little or no prior experience with software debugging or machine learning, we did not accept participants who had more programming experience than an introductory-level course. A total of 37 participants experienced the control condition and 40 participants took part in the treatment condition.

A researcher introduced participants to the prototype via a brief hands-on tutorial that explained how to use it, but did not discuss how it made predictions. Participants then had three minutes to explore the prototype on their own. To avoid learning effects, the tutorial and practice session involved messages about topics (*outer space* and *medicine*) different from the main experiment task.

The main experiment task followed the practice session. Participants were asked to "make the computer's predictions as accurate as possible" and given 30 minutes to work. The software logged all participant interactions and logged evaluations of its internal classifier at 30-second intervals.

After the main task concluded, we assessed participants' mental models via a questionnaire. This test instrument evaluated how well participants understood the two components that contribute to the MNB classifier's predictions: feature presence and class ratios. Because feature presence can be easy to detect given certain words (e.g., the word "hockey" is obviously related to the concept of *hockey*), we evaluated participants' understanding of feature presence using both "obvious" and "subtle" features. We define "subtle" features as words that are not normally associated with a topic, but appear in the classifier's training set and thus will impact classification. Participants were given three short messages about two topics (*swimming* and *tennis*) and told that "these are the only messages the software has learned from". Participants in the treatment condition were also given an *Important words* explanation similar to the one they saw in the prototype. A second sheet displayed 12 messages, each only one or two

sentences long, and asked participants which topic the classifier would assign to each message, and why. The messages were constructed such that only one component—either an obvious feature, a subtle feature, or class ratios—was entirely responsible for the classification.

To understand participants' reactions to EluciDebug, the post-task questionnaire also asked participants about various features of the prototype and their perceived task load during the experiment (via the NASA-TLX questionnaire [14]).

Data analysis

We used non-parametric methods for all statistical analyses. As suggested in [29], we used Mann–Whitney U-tests for ordinal data, Wilcoxon signed rank tests for interval data, and Spearman's ρ for correlations.

To analyze participants' mental models, a researcher graded participant responses to the post-task mental model questionnaires. Because some participants may have randomly guessed which topic the classifier would predict for each message, we ignored all predicted topics and only graded the reason participants' stated for the classifier's prediction. Participants earned two points for correct reasons and one point for partially correct reasons. The researcher graded participant responses without knowing which condition the participant was in (i.e., blindly). Each participants' points were summed to yield a mental model score with a maximum possible value of 24.

We analyzed classifier performance via the F_1 *score*. This combines two simpler measures, *precision* and *recall*, each of which can range from 0 to 1. In the context of a binary classification system that predicts whether each input is positive or negative, a precision of 0 indicates that none of its positive predictions were correct, while a precision of 1 indicates that all of its positive predictions were correct. For the same system, a recall of 0 indicates the classifier did not correctly identify any of the positive items, while a recall of 1 indicates that it correctly identified all of them. As the harmonic mean of precision and recall, F_1 also ranges from 0 (no precision and no recall) to 1 (perfect precision and recall).

We supplemented our evaluation of classifier accuracy with an additional offline experiment using a separate feature selection method. Recall that EluciDebug limits its classifier to the 10 features with the highest information gain. Text classifiers, however, often include most—if not all—of the words in the training set as features. Thus, we analyzed participants' classifiers using both *HighIG* features and *Comprehensive* features. For control participants (who could not provide feature-based feedback), *HighIG* was recomputed after each message was labeled and kept the 10 highest information gain features. For treatment participants, *HighIG* was never recomputed; instead, participants needed to modify it manually by adding, removing, or adjusting features. The *Comprehensive* feature set included all words from the set of labeled messages at the end of the experiment. The classifiers participants interacted with used the *HighIG* features; the *Comprehensive* features were only used for offline analysis.

RESULTS

Explaining Corrections to EluciDebug

EluciDebug includes several methods for users to explain corrections to the classifier, and treatment participants made frequent use of all of them. On average, they added 34.5 new features, removed 8.2 features (including 7.4 of EluciDebug's 10 initial features), and made 18.3 feature adjustments (e.g., increasing or decreasing a feature's importance to a topic). Control participants—who could not provide feature-based feedback—instead relied on instance-based feedback to adjust EluciDebug's predictions, labeling an average of 182 messages vs. the treatment average of 47 ($W = 1395$, $p < .001$). Control participants also examined more messages, averaging 296 message views vs. 151 views ($W = 1306$, $p < .001$). Treatment participants thus provided less feedback overall (and needed to explore less of the dataset while providing it), instead leveraging EluciDebug's abilities to target their feedback at features rather than instances.

This feature-based feedback proved efficient at improving participants' classifiers. We examined the change in F_1 for each participant's classifier during the experiment and divided this by the number of actions the participant made that could influence the classifier's predictions (instances labeled and features added, removed, or adjusted). The results, shown in Figure 6 (left), were that treatment participants performed fewer actions, but each of their actions resulted in larger classifier improvements than those of control participants. Treatment participants' feedback was twice as efficient as control participants' using *HighIG* features (0.16% vs. 0.34% F_1 improvement per action, $W = 207$, $p < .001$), and remained superior when using the *Comprehensive* feature set (0.65% vs. 0.97%, $W = 367$, $p < .001$).

We thus have evidence that when users can only provide a limited amount of feedback to a learning system (such as when labeling instances is expensive, insufficient instances are available for labeling, or the user's time is constrained), Explanatory Debugging can result in superior classifiers than a traditional black-box instance labeling approach. Indeed, Figure 6 (right) shows that by the end of the 30-minute experiment, treatment participants had created classifiers that were roughly 10% more accurate than control participants, averaging F_1 scores of 0.85 vs. 0.77 ($W = 237$, $p < .001$).

However, our analysis with the *Comprehensive* feature set suggests that when the user can label many instances, instance labeling with a large feature set may be preferable to Explanatory Debugging—at least to initially train a classifier. The combination of a large training set and many features allowed control participants' classifiers to edge out those of treatment participants by about 8% (Figure 6, right). Even though treatment participants' feedback was up to twice as efficient, control participants provided almost four times as many labeled instances, allowing them to train classifiers with an average F_1 of 0.94, while treatment participants averaged 0.87 ($W = 1290$, $p < .001$).

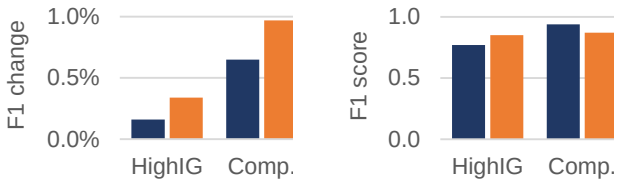

Figure 6. (Left) Average classifier F_1 improvement per user action for control (dark blue) and treatment (light orange); treatment participants controlled their classifiers up to twice as efficiently as control participants. (Right) Average classifier F_1 scores per condition. Control participants needed four times as much data and the *Comprehensive* feature set to create better classifiers than treatment participants.

To verify it was the amount of instance-based feedback that allowed control participants to outperform treatment participants when *Comprehensive* features were considered, we analyzed the accuracy of their classifiers after the same number of actions had been performed. Figure 7 shows the F_1 scores after n feedback actions using the *HighIG* (solid line) and *Comprehensive* (dotted line) feature sets. Given the same number of actions, control participants never outperformed treatment participants. This suggests that when treatment participants did provide instance-based feedback (which was the only type of feedback used for the *Comprehensive* analysis), it was usually more useful than control participants' feedback.

We also analyzed participant reactions to the two prototype variations. Treatment participants liked their variant more than control participants, rating its helpfulness as 4.8 vs. 4.3 on a 6-point scale ($W = 474$, $p = .006$). Further, we found no evidence that treatment participants felt Explanatory Debugging involved more work than black-box instance labeling. We used the NASA-TLX survey to measure participants' perceived task load while attempting to improve their classifier, but found no evidence of a difference between conditions.

These classifier measures reveal three findings. First, in situations where large amounts of training data is unavailable or expensive to obtain, Explanatory Debugging (as instantiated in EluciDebug) allows users to successfully train a classifier by telling it about features instead of instances. Second, the mix of feature- and instance-based feedback provided by treatment participants was more efficient than the purely instance-based feedback provided by control participants, suggesting that when an end user has a specific goal in mind (such as our Alice example from earlier), Explanatory Debugging can help the user quickly realize their goal.

Third, control participants' success with using large amounts of instance-based feedback suggests that in domains where labeling instances is quick and practical, some combination of feature- and instance-based feedback may be best. In fact, such systems may need to emphasize the potential usefulness of labeling instances. In our experiment, the mere presence of feature-based feedback tools appears to have biased participants against instance-based feedback: 3 treatment participants did not provide any at all, while the smallest number of labeled

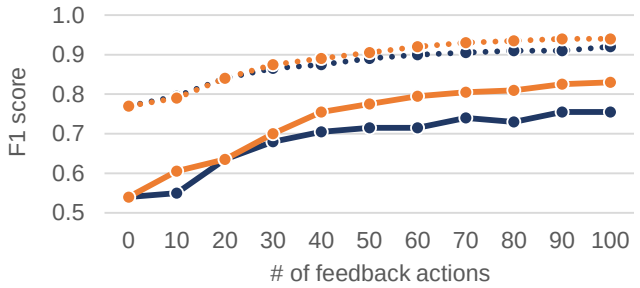

Figure 7. Treatment participants (light orange) created equivalent or better classifiers than control participants (dark blue) using the same amount of feedback. This held for both the *HighIG* (solid) and *Comprehensive* (dotted) feature sets.

instances from a control participant was 56—more than even the treatment *average* of 47 labeled instances.

EluciDebug's Explanations to End Users

Before users can correct a machine learning system's explanation of its reasoning, they must first *understand* the explanation. This understanding is reflected in their mental model.

Treatment participants built significantly better mental models than participants in the control condition. As shown in Table 1, treatment participants scored 52% higher on the mental model assessment than control participants ($W = 259$, $p < .001$). Much of this difference stems from treatment participants identifying *all* of the keywords the classifier used, while control participants often identified only the "obvious" keywords. In fact, treatment participants averaged a score of 14.1 out of 16 (88%) during the keyword assessment, suggesting they firmly understood how the classifier involved keywords—regardless of whether the words had any semantic association with their topics—in its reasoning.

Table 1 also suggests treatment participants may have better understood that the classifier used class ratios as part of its prediction strategy than participants in the control condition ($W = 619.5$, $p = .099$), but the evidence is weak—even among treatment participants, the mean score was only 1.8 out of 8. Further, a majority of participants in both conditions failed to answer *any* class ratio question correctly, suggesting that this explanation either failed to convey relevant information about how class ratios were used by the classifier, or failed to attract participants' attention.

In general, however, control participants wanted the same information available to the treatment group. As one participant stated:

> *C1: "More information on how emails are sorted would help the user target emails to categorize, which would increase accuracy."*

Another control participant (C15) described the software as *"annoying"*, but that working with it *"would have been easier if we knew how it made predictions"*, while still another (C4) said it was annoying to work with because he *"didn't know what it was basing its predictions off of"*. A fourth participant (C11) even asked for a similar feedback mechanism as was

Model component	Max score	Control mean (SD)	Treatment mean (SD)	*p*-value
Obvious features	8	6.7 (2.7)	7.3 (1.8)	.345
Subtle features	8	2.8 (2.6)	**6.8 (1.9)**	<.001
Class ratios	8	0.6 (1.5)	1.8 (3.0)	.099
Total score	24	10.4 (5.3)	**15.8 (4.6)**	<.001

Table 1. Treatment participants finished the experiment with significantly higher mental model scores than control participants.

available to treatment participants, saying the software was *"time-consuming"* because there was *"no way to highlight key words/terms"*. A fifth control participant summed it up:

C30: "That was a long 30 minutes."

Participants in the treatment condition, conversely, appreciated EluciDebug's explanations and feedback mechanisms:

T24: "Not difficult to understand/operate, doesn't take a lot of effort."

T40: "It was really fast to get a high degree of accuracy."

T37: "I felt in control of all the settings."

T6: "It was so simple my parents could use it."

Overall, our principled Explanatory Debugging approach successfully helped participants develop accurate mental models of the classifier they used, and participants benefited from this additional knowledge. Spearman's ρ confirms a link between participants' mental model scores and their classifier's F_1 scores ($\rho[75] = .282$, $p = .013$)

DISCUSSION

RQ1 and RQ2: Efficient and Accurate Personalization

Our results suggest that Explanatory Debugging can be an *efficient* method for users to personalize a machine learning system, but that it may not always result in the most *accurate* classifiers. For example, we found that feature-based feedback was up to twice as effective as instance-based feedback, but instance labeling could still yield more accurate classifiers given enough labels and features (in our experiment, four times as many labels were needed). In situations where labeling instances is considerably easier or faster than providing feature-based feedback, users may be better served by labeling a large number of instances than a small number of features.

However, when users need fine-grained control over a classifier, Explanatory Debugging has two advantages beyond efficiency. First, it does not require that training data exist, and thus can be used to bootstrap a learning system. Even a trained system that suddenly needs to support a new output type may benefit from such bootstrapping. Second, the quick improvements—during even the first 10 user actions—treatment participants made to their classifiers suggest that users will remain engaged with Explanatory Debugging. This matters because research has shown that if an end-user debugging technique is not perceived as useful after a small number of interactions, users are unlikely to continue using it [32]. Seeing an immediate improvement after providing feedback suggests that users will continue to view and correct the Explanatory Debugging explanations, while the lack of such an improvement may discourage users of black-box instance labeling systems from continuing to provide feedback.

RQ3: Mental Models

Not only did Explanatory Debugging help participants build useful mental models, it accomplished this without a perceived increase in task load. We found no evidence that treatment participants found the extra information or feedback mechanisms more difficult to understand or use; instead, treatment participants' responses suggest they appreciated having such information available.

Indeed, the fact that many control participants' requested explanations remarkably similar to those the treatment participants saw suggests the need for machine learning systems to be able to explain their reasoning in accordance with Explanatory Debugging's *Explainability* principle. Even if this information is hidden by default, users should be able to view such explanations on demand. Further, because machine learning systems are meant to classify items as their user would, the user must have some method to correct the system's mistakes. Thus, we hypothesize that including Explanatory Debugging-style explanations without also supporting our *Correctibility* principle will frustrate users—they would be able to see what needs correcting, but without a clear mapping to the actions they need to take.

CONCLUSION

Overall, Explanatory Debugging's cycle of explanations—from the learning system to the user, and from the user back to the system—resulted in smarter users and smarter learning systems. Participants using Explanatory Debugging understood how the learning system operated about 50% better than control participants, and this improvement correlated with the F_1 scores of participants' classifiers. Each piece of feedback provided by Explanatory Debugging participants was worth roughly two pieces of feedback provided by control participants; even when we expanded our analysis to include a comprehensive feature set, treatment participants still maintained a 50% edge over the control group. Further, participants liked Explanatory Debugging, rating this variant of EluciDebug higher than the control group and responding enthusiastically to the system's explanations.

Our results show that when end users want to personalize a machine learning system, Explanatory Debugging is a more controllable and satisfying approach than black-box instance labeling. The approach's focus on users—in which the system explains its reasoning and the user explains back corrections as needed—enables ordinary end users to get the most out of the learning system on which they are starting to depend.

ACKNOWLEDGMENTS

[Anonymized for blind review.]

REFERENCES

1. Amershi, S., Cakmak, M., Knox, W. B., and Kulesza, T. Power to the people: The role of humans in interactive machine learning. *AI Magazine* (in press).

2. Amershi, S., Fogarty, J., Kapoor, A., and Tan, D. Examining multiple potential models in end-user interactive concept learning. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2010), 1357–1360.
3. Blackwell, A. F. First steps in programming: A rationale for attention investment models. In *Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, IEEE (2002), 2–10.
4. Bostandjiev, S., O'Donovan, J., and Höllerer, T. TasteWeights: A visual interactive hybrid recommender system. In *Proceedings of the 6th ACM Conference on Recommender Systems* (2012), 35–42.
5. Bryan, N. J., Mysore, G. J., and Wang, G. ISSE: An interactive source separation editor. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2014), 257–266.
6. Bunt, A., Lount, M., and Lauzon, C. Are explanations always important? A study of deployed, low-cost intelligent interactive systems. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces* (2012), 169–178.
7. Cakmak, M., Chao, C., and Thomaz, A. L. Designing interactions for robot active learners. *IEEE Transactions on Autonomous Mental Development 2*, 2 (2010), 108–118.
8. Cramer, H., Evers, V., Ramlal, S., van Someren, M., Rutledge, L., Stash, N., Aroyo, L., and Wielinga, B. The effects of transparency on trust in and acceptance of a content-based art recommender. *User Modeling and User-Adapted Interaction 18*, 5 (2008), 455–496.
9. Craven, M. W., and Shavlik, J. W. Using neural networks for data mining. *Future generation computer systems 13* (1997), 211–229.
10. Dzindolet, M. T., Peterson, S. A., Pomranky, R. A., Pierce, L. G., and Beck, H. P. The role of trust in automation reliance. *International Journal of Human-Computer Studies 58*, 6 (2003), 697–718.
11. Fails, J. A., Olsen, D. R., and Jr. Interactive machine learning. In *Proceedings of the 8th International Conference on Intelligent User Interfaces* (2003), 39–45.
12. Fiebrink, R., Cook, P. R., and Trueman, D. Human model evaluation in interactive supervised learning. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2011), 147–156.
13. Fogarty, J., Tan, D., Kapoor, A., and Winder, S. CueFlik: Interactive concept learning in image search. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2008), 29–38.
14. Hart, S. G., and Staveland, L. E. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Advances in Psychology 52* (1988), 139–183.
15. Herlocker, J. L., Konstan, J. A., and Riedl, J. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work* (2000), 241–250.
16. Johnson-Laird, P. N. *Mental models: Towards a cognitive science of language, inference, and consciousness*. Harvard University Press, 1983.
17. Kapoor, A., Lee, B., Tan, D., and Horvitz, E. Interactive optimization for steering machine classification. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2010), 1343–1352.
18. Kempton, W. Two theories of home heat control. *Cognitive Science 10* (1986), 75–90.
19. Kibriya, A. M., Frank, E., Pfahringer, B., and Holmes, G. Multinomial naive Bayes for text categorization revisited. In *AI 2004: Advances in Artificial Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, 488–499.
20. Knox, W. B., and Stone, P. Reinforcement learning from human reward: Discounting in episodic tasks. In *Proceedings of the 21st IEEE International Symposium on Robot and Human Interactive Communication* (2012), 878–885.
21. Kulesza, T., Stumpf, S., Burnett, M. M., and Kwan, I. Tell me more? The effects of mental model soundness on personalizing an intelligent agent. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2012), 1–10.
22. Kulesza, T., Stumpf, S., Burnett, M. M., Wong, W.-K., Riche, Y., Moore, T., Oberst, I., Shinsel, A., and McIntosh, K. Explanatory debugging: Supporting end-user debugging of machine-learned programs. In *Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing* (2010), 41–48.
23. Kulesza, T., Stumpf, S., Burnett, M. M., and Yang, S. Too much, too little, or just right? Ways explanations impact end users' mental models. *Proceedings of the 2013 IEEE Symposium on Visual Languages and Human-Centric Computing* (2013), 3–10.
24. Kulesza, T., Stumpf, S., Wong, W.-K., Burnett, M. M., Perona, S., Ko, A. J., and Oberst, I. Why-oriented end-user debugging of naive Bayes text classification. *ACM Transactions on Interactive Intelligent Systems 1*, 1 (2011).
25. Lacave, C., and Díez, F. J. A review of explanation methods for Bayesian networks. *The Knowledge Engineering Review 17*, 2 (2002), 107–127.
26. Lim, B. Y. *Improving understanding and trust with intelligibility in context-aware applications*. PhD thesis, Carnegie Mellon University, 2012.
27. Lim, B. Y., and Dey, A. K. Assessing demand for intelligibility in context-aware applications. In *Proceedings of the 11th International Conference on Ubiquitous Computing* (2009), 195–204.
28. Lim, B. Y., Dey, A. K., and Avrahami, D. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2009), 2119–2128.
29. McCrum-Gardner, E. Which is the correct statistical test to use? *British Journal of Oral and Maxillofacial Surgery*

46 (2008), 38–41.

30. Norman, D. A. Some observations on mental models. In *Human-Computer Interaction*, R. M. Baecker and W. A. S. Buxton, Eds. San Francisco, CA, USA, 1987, 241–244.
31. Norman, D. A. *The Design of Everyday Things*. Revised and Expanded Edition. Basic Books, 2002.
32. Prabhakararao, S., Cook, C., Ruthruff, J., Creswick, E., Main, M., Durham, M., and Burnett, M. M. Strategies and behaviors of end-user programmers with interactive fault localization. In *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments* (2003), 15–22.
33. Rowe, M. B. *Teaching science as continuous inquiry*. McGraw-Hill, 1973.
34. Settles, B. Active learning literature survey. Tech. Rep. 1648, University of Wisconsin-Madison, 2010.
35. Stumpf, S., Rajaram, V., Li, L., Wong, W.-K., Burnett, M. M., Dietterich, T., Sullivan, E., and Herlocker, J. Interacting meaningfully with machine learning systems: Three experiments. *International Journal of Human-Computer Studies 67*, 8 (2009), 639–662.
36. Szafron, D., Greiner, R., Lu, P., and Wishart, D. Explaining naïve Bayes classifications. Tech. Rep. TR03-09, University of Alberta, 2003.
37. Talbot, J., Lee, B., Kapoor, A., and Tan, D. S. EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2009), 1283–1292.
38. Thomaz, A. L., and Breazeal, C. Transparency and socially guided machine learning. In *Proceedings of the 5th International Conference on Development and Learning* (2006).
39. Tintarev, N., and Masthoff, J. Evaluating the effectiveness of explanations for recommender systems. *User Modeling and User-Adapted Interaction 22*, 4-5 (2012), 399–439.
40. Tullio, J., Dey, A. K., Chalecki, J., and Fogarty, J. How it works: A field study of non-technical users interacting with an intelligent system. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (2007), 31–40.
41. van der Meij, H., and Carroll, J. M. Principles and heuristics for designing minimalist instruction. In *Minimalism beyond the Nurnberg funnel*, J. M. Carroll, Ed. MIT Press, Cambridge, MA, 1998, 19–53.
42. Vig, J., Sen, S., and Riedl, J. Navigating the tag genome. In *Proceedings of the 16th International Conference on Intelligent User Interfaces* (2011), 93–102.
43. Yang, R., and Newman, M. W. Learning from a learning thermostat: Lessons for intelligent systems for the home. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (2013), 93–102.
44. Yang, Y., and Pedersen, J. O. A comparative study on feature selection in text categorization. In *Proceedings of the Twentieth International Conference on Machine Learning* (1997), 412–420.