Why-Oriented End-User Debugging of Naive Bayes Text Classification

TODD KULESZA, Oregon State University
SIMONE STUMPF, City University London
WENG-KEEN WONG, MARGARET M. BURNETT,
and STEPHEN PERONA, Oregon State University
AMY J. KO, University of Washington
IAN OBERST, Oregon State University

Machine learning techniques are increasingly used in *intelligent assistants*, that is, software targeted at and continuously adapting to assist end users with email, shopping, and other tasks. Examples include desktop SPAM filters, recommender systems, and handwriting recognition. Fixing such intelligent assistants when they learn incorrect behavior, however, has received only limited attention. To directly support end-user "debugging" of assistant behaviors learned via statistical machine learning, we present a Why-oriented approach which allows users to ask questions about how the assistant made its predictions, provides answers to these "why" questions, and allows users to interactively *change* these answers to debug the assistant's current and future predictions. To understand the strengths and weaknesses of this approach, we then conducted an exploratory study to investigate barriers that participants could encounter when debugging an intelligent assistant using our approach, and the information those participants requested to overcome these barriers. To help ensure the inclusiveness of our approach, we also explored how gender differences played a role in understanding barriers and information needs. We then used these results to consider opportunities for Why-oriented approaches to address user barriers and information needs.

Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms: Design, Experimentation, Human Factors, Performance

Additional Key Words and Phrases: Debugging, end-user programming, machine learning

ACM Reference Format:
Kulesza, T., Stumpf, S., Wong, W.-K., Burnett, M. M., Perona, S., Ko, A., and Oberst, I. 2011. Why-oriented end-user debugging of naive Bayes text classification. ACM Trans. Interact. Intell. Syst. 1, 1, Article 2 (October 2011), 31 pages.
DOI = 10.1145/2030365.2030367 http://doi.acm.org/10.1145/2030365.2030367

1. INTRODUCTION

Machine learning is increasingly used to power *intelligent assistants*, that is, software that continuously tunes its behavior to match a specific end user's interaction patterns, so as to assist the user in getting work done (helping organize email, recommending books relevant to the user's interests, typing hand-written notes, and so on). When

This work was supported in part by NSF grant IIS-0803487 and NSF grant IIS-0917366.
Authors' addresses: T. Kulesza, School of Electrical Engineering and Computer Science, Oregon State University; email: kuleszto@eecs.oregonstate.edu; S. Stumpf, Centre for Human Computer Interaction Design, City University London; W.-K. Wong, M. M. Burnett, and S. Perona, School of Electrical Engineering and Computer Science, Oregon State University; A. Ko, The Information School, University of Washington; I. Oberst, School of Electrical Engineering and Computer Science, Oregon State University.

DOI 10.1145/2030365.2030367 http://doi.acm.org/10.1145/2030365.2030367

assistants adapt themselves in helpful ways, the benefits are obvious: SPAM filters misclassify fewer emails, recommendation systems find items of interest quickly, and handwriting-recognition tools do a good job typing up the user's notes.

What should a user do, however, when the assistant adapts itself in unhelpful ways? A straightforward answer might seem to be "provide more training examples," but gathering sufficient training examples can be unrealistically time-consuming for an end user, and may not solve the particular misbehaviors the user cares most about. We posit instead that, just as with regular software, when an intelligent assistant fails, it should be possible to debug it directly. Further, since intelligent assistants do not return to the hands of programming specialists, the only people in a position to debug intelligent assistants are the end users themselves—they are the only ones using that specific adaptation.

Enabling end users to debug their assistants is nontrivial: most users have little knowledge of how systems based on machine learning operate. Prior work, however, has shown that end users can learn how an intelligent assistant makes its decisions [Tullio et al. 2007]. We therefore prototyped a new approach that aims to support end users in guiding and correcting—that is, debugging—automated text classification. The domain we chose to explore was classification of email messages into email folders, as supported by the widely used naive Bayes algorithm.

Our work makes three primary contributions. First, we present a new *Why-oriented approach* to allow end users to debug their intelligent assistants. The approach not only focuses on answering end-user questions about why the assistant is behaving in its current manner; it also provides interactive explanations that serve as a debugging mechanism. Because end users are directly debugging the assistant (as opposed to providing new training examples), our approach presents information that a programmer would expect when directly debugging other kinds of software: representations of both the intelligent assistant's *logic* (analogous to its source code) and *runtime state*. The essence of the approach is simple: if the users do not like the explanations that answer their "why" questions, they can change the explanations to direct the assistant to behave more accurately.

Second, our work explores the difficulties experienced by end users attempting to debug an assistant in this Why-oriented manner. We present an exploratory study that identifies and categorizes the barriers users encountered while debugging using a prototype based on our interactive approach. We also investigated both what information needs end users expressed while debugging, and when they needed this information. Finally, because of recent evidence of gender differences in debugging (e.g., Grigoreanu et al. [2008] and Subrahmaniyan et al. [2008]), we investigated differences among the barriers and information needs by gender to highlight design considerations that could differently impact these subsets of end users. Our goal was to uncover barriers posing obstacles to users of our why-oriented approach. Thus, our user study and resulting analysis focused on the barriers users encountered and information needs users expressed, rather than an empirical evaluation of the prototype's design. Third, we consider how to progress further with Why-oriented debugging by identifying machine learning solutions and open research questions as to how machine learning systems can address the information needs and barriers of end users attempting to debug their intelligent assistants.

2. RELATED WORK

As part of its support for debugging, our Why-oriented approach explains the logic of an intelligent assistant to end users so that they can understand how to debug it. There are thus two major classes of prior work: techniques that support end-user debugging, and techniques that support communication between users and machine learning systems. We next discuss each, identifying the foundations that informed the design of our own prototype.

2.1. End-User Debugging

Outside of machine learning, there are a number of debugging systems that help end users find and understand the causes of faulty behavior. For example, in the spreadsheet domain, WYSIWYT [Burnett et al. 2003] has a fault-localization device that helps users reason about successful and unsuccessful “tests” to locate cells whose formulas are likely to be faulty. Woodstein [Wagner and Lieberman 2004] helps users to debug e-commerce problems by explaining events and transactions between services.

The Whyline [Ko 2008] pioneered a method (also outside the realm of machine learning) to debug certain types of programs in an explanation-centric way. Because the Whyline informs our approach, we present it in some detail.

The original Whyline was explored in three contexts: event-based virtual worlds written in the Alice programming system [Ko et al. 2004], Java programs [Ko et al. 2008], and the Crystal system for debugging unexpected behaviors in complex interfaces [Myers et al. 2006]. In each case, the tools help programmers understand the causes of program output by allowing them to select an element of the program and receive a list of why and why not questions and answers in response. These “Why?” questions and answers are extracted automatically from a program execution history, and “Why not” answers derive from a reachability analysis to identify decision points in the program that could have led to the desired output. In the Crystal prototype, rather than presenting answers as sequences of statement executions, answers are presented in terms of the user-modifiable input that influenced code execution. In all of these Whyline tools, the key design idea is that users select some output they want to understand, and the system explains the underlying program logic that caused it.

Some researchers have focused on the problems and needs that end users encounter when they debug. For example, Ko et al. explored *learning barriers* that novice programmers encountered when learning how to solve problems in a programming environment [Ko et al. 2004]. Researchers from the WYSIWYT project have categorized the information needs of end users in debugging spreadsheets [Kissinger et al. 2006], enumerating the types of information that end users sought. These barriers and information needs may also relate to nontraditional debugging, such as fixing intelligent assistants.

2.2. Communicating with Machine Learning Systems

Several recent studies have highlighted the need to explain a machine learning algorithm’s reasoning to users. For example, Patel et al. examined obstacles faced by developers familiar with machine learning who need to apply machine learning to real-world problems [Patel et al. 2008]. Glass et al. investigated the types of questions users familiar with intelligent agents would like to ask an adaptive agent in order to increase their trust in the agent [Glass et al. 2008]. Similarly, Lim and Dey identified the types of information end users wanted context-aware applications to provide when explaining their current context, to increase both trust in and understanding of the system [Lim and Dey 2009].

Much of the work in explaining probabilistic machine learning algorithms has focused on the naive Bayes classifier [Becker et al. 2001; Kononenko 1993] and, more generally, on linear additive classifiers [Poulin et al. 2006]. Explanation of these algorithms is relatively straightforward and computationally feasible on modern hardware. Tullio et al. reported that, given some basic types of explanations, end users can understand how machine learning systems operate [Tullio et al. 2007], with the caveat that overcoming any preliminary faulty assumptions may be problematic. More sophisticated, though computationally expensive, explanation algorithms have been developed for general Bayesian networks [Lacave and Diez 2002]. Finally, Lim et al. [2009] investigated the usefulness of the Whyline approach for explaining simple decision trees, and found that the approach was viable for explaining this relatively understandable form of machine learning.

Regarding allowing end users to actually influence behaviors in machine learning settings, some Programming by Demonstration (PBD) systems learn programs interactively, via machine learning techniques based on sequences of user actions (see [Lieberman 2001] for a collection of such systems). When debugging these kinds of programs, end-user corrections are limited to the addition or removal of training data—unless the user reverts to a traditional programming language such as Lisp (e.g., Vander Zanden and Myers [1995]). For example, Gamut allows users to "nudge" the system when it makes a mistake, leading to the addition or deletion of training examples [McDaniel and Myers 1999]. Recent work with PBD systems allows some debugging of programs [Chen and Weld 2008], but their technique only allows the user to retract actions in a demonstration, which results in adding missing values to the training data rather than directly modifying the classifier's logic. Still other systems allow users to patch up specific mistakes by an intelligent assistant, but do not take these corrections into account when the assistant makes future decisions. For example, if CoScripter/Koala programs misidentify web page objects, the user can specify the correct object for a particular page; the fix, however, will not affect how the program identifies similar objects on different pages [Little et al. 2007].

Debugging, that is, directly changing the logic of intelligent assistants, has received only limited attention. One such system, EnsembleMatrix [Talbot et al. 2009], provides both a visualization of a classifier's accuracy and the means to adjust its logic. EnsembleMatrix, however, is targeted at machine-learning experts developing complex ensemble classifiers, rather than end users working with the resulting classifiers. ManiMatrix [Kapoor et al. 2010] provides an interactive visualization of a classifier's accuracy, but user interactions are restricted to the modification of a classifier's cost matrix.

Our own prior research has begun to explore end-user interactions with intelligent assistants, to understand how to effectively enable end users to debug such programs. Using a paper prototype, we previously investigated three different types of explanations (keyword-based, rule-based, and similarity-based) that machine learning systems could provide to end users regarding why it was behaving in a particular manner, as well as user reactions to these explanations [Stumpf et al. 2007]. This paper prototype was also used to elicit corrections to the logic from participants (e.g., adjusting feature weights), allowing us to design an interactive prototype supporting the explanations best understood by participants and the types of corrections they most requested [Stumpf et al. 2008]. The interactive prototype permitted us to run offline experiments studying the effects of the corrections provided by end users on prediction accuracy versus traditional label-based corrections. The results suggest that even when very simple corrections are incorporated into the assistant's decision-making process, it has the potential to increase the accuracy of the resulting predictions [Stumpf et al. 2008, 2009]. For some users, however, the quality of the assistant's predictions decreased as a result of their corrections; there were barriers that prevented them from successfully debugging the assistant.

In summary, the ability of end users to interactively debug machine-learned logic has been limited. Researchers have begun to investigate how such logic can be explained to end users, but user corrections, if available at all, have been heavily restricted to specific forms (e.g., addition of training data) or situations (e.g., the initial creation of a new program via PBD). In addition to explaining the underlying logic, this article also addresses supporting end users actually fixing the logic of an intelligent assistant.

3. A WHY-ORIENTED APPROACH FOR DEBUGGING INTELLIGENT ASSISTANTS

Inspired by the success of the Whyline's support for debugging [Ko 2008; Myers et al. 2006] and favorable user feedback regarding "why" and "why not"-style explanations [Lim et al. 2009; Lim and Dey 2009], we have developed a Why-oriented approach for end-user debugging of intelligent assistants. Our prototype of the approach allows end users to debug an assistant's predicted folder classifications of an email application.

Our approach is the first to combine the following elements.

(1) It allows end users to directly ask questions of statistical machine learning systems (e.g., "why will this message be filed to 'Systems'?");
(2) The answers explain both the current logic and execution state (e.g., a visualization that shows the importance of features/words to this folder, and how certain the system is that it belongs in this folder);
(3) Users can change these explanations by direct manipulation to debug the system's logic. These changes result in real-time adjustments to the assistant's logic and the resulting predictions.

3.1. Design of Why Questions

Developing the set of why questions about the logic and execution state of the intelligent assistant comprised two stages. In the Generate stage, we generated the set of all possible questions that could be asked via the creation of a formal grammar. Then, in the Filter stage, we filtered the set of questions generated by the grammar to remove impossible questions, where *impossible* refers to situations that were not related to or achievable by debugging an intelligent assistant. We chose this generate-and-filter approach, as opposed to handcrafting the questions ourselves, to be sure we would not miss any possible questions.

We began the Generate stage by inventorying the domain objects, such as messages and folders, which inform the assistant's logic. We also inventoried, at a more abstract level, all logic (and corresponding run-time states) feeding into the predictions, such as the importance of a word to a folder. Finally, we inventoried the types of feedback the assistant could provide about its logic and execution state, such as its folder predictions and its estimation of word importance. The first inventory (domain objects) is enumerated in the *Subjects* section of Table I, and the latter two together are the *Situations* section of Table I which generates a super-set of these types of question-word phrases. Finally, we added the *Modifiers* section to allow generated questions to be more specific (e.g., a question about recently changed predictions, rather than *all* changed predictions). The *Queries* section of Table I describes how the various components may be combined to form the universe of possible questions.

For example, our grammar defines a possible query as "[*question*] [*verb*] [*subject*] [*situation*]". Selecting the first listed production for [*question*], [*verb*], and [*subject*] produces the production "Why will this message [*situation*]?" The first production for the *Situations* component (*current classification*) can then be substituted for [*situation*], resulting in the completed question, "Why will this message be filed to Enron News?"

In the Filter stage, we filtered the questions generated by the grammar in three ways. First, we removed questions about subject/situation combinations that could not be accomplished by the intelligent assistant. For example, a message (a *Subject*) can change its current classification (a *Situation*), but words and folders (also *Subjects*) cannot. We further refined our list of questions by removing any questions not relating to debugging, such as "Why can important words be displayed?" Finally, we removed rephrasings of similar questions; this explains why there are no questions beginning with "How can. . . ", since asking "How can I make something happen?" can usually be answered by the same information provided by the "Why didn't something happen?"

Table I. Query Grammar to Generate Our Why Questions

Components	Productions
Questions	• Why . . .? • How . . .?
Verbs	• To be • To do • Can/Make
Modifiers	• This • All • Many • Recent • Important
Subjects	• Message • Folder • Word • Change • End user (i.e., "I")
Situations	• Current classification • Change in classification • Importance • Availability (i.e., displayed in the UI)
Queries	• [query] = [question] [verb] [subject] [situation] | [question] [situation] [verb] [subject] • [situation] = [situation] | [situation] [subject] | [subject] [situation] • [subject] = [subject] | [modifier] [subject] • [modifier] = [modifier] | [modifier] [modifier]

Table II. Why Questions and Query Grammar Productions to Generate Them. (Superscript text is used to map each component to the English phrase it generated (q = question, v = verb, m = modifier, su = subject, si = situation). Underlined text represents a component consisting of multiple words, and text in <brackets> is dynamically replaced with the word or folder the user has selected)

Why Questions	Generating Production
Why^{q} will^{v} this^{m} message^{su} be filed to^{si} <Personal>^{su}?	[question] [verb] [modifier] [subject] [situation] [subject]
Why^{q} won't^{v} this^{m} message^{su} be filed to^{si} <Bankruptcy>^{su}?	[question] [verb] [modifier] [subject] [situation] [subject]
Why^{q} did^{v} this^{m} message^{su} turn red^{si}?	[question] [verb] [modifier] [subject] [situation]
Why^{q} wasn't^{v} this^{m} message^{su} affected by^{si} my recent^{m} changes^{su}?	[question] [verb] [modifier] [subject] [situation] [modifier] [subject]
Why^{q} did^{v} so many^{m} messages^{su} turn red^{si}?	[question] [verb] [modifier] [subject] [situation]
Why^{q} is^{v} this^{m} email^{su} undecided^{si}?	[question] [verb] [modifier] [subject] [situation]
Why^{q} does^{v} <banking>^{su} matter to^{si} the <Bankruptcy> folder^{su}?	[question] [verb] [subject] [situation] [subject]
Why^{q} aren't^{v} all important^{m} words^{su} shown^{si}?	[question] [verb] [modifier] [subject] [situation]
Why^{q} can't make^{v} I^{su} this^{m} message^{su} go to^{si} <Systems>^{su}?[1]	[question] [verb] [subject] [modifier] [subject] [situation] [subject]

questions. This decision is consistent with the results of Lim et al. [2009], who found that participants were more successful in reasoning about intelligent assistants (in the form of decision trees) when presented with answers to either "Why. . . " or "Why not. . . " questions, than participants were when presented with answers to "How to. . . " questions.

The Filter stage resulted in the nine Why questions depicted in Table II. (While this number may seem small, the original Whyline required only six types of questions [Ko 2008] in the complex domain of Java programming.) Figure 1 shows how these questions were presented in the prototype, including their context dependency (such as the ability of the user to select questions relating to specific folders).

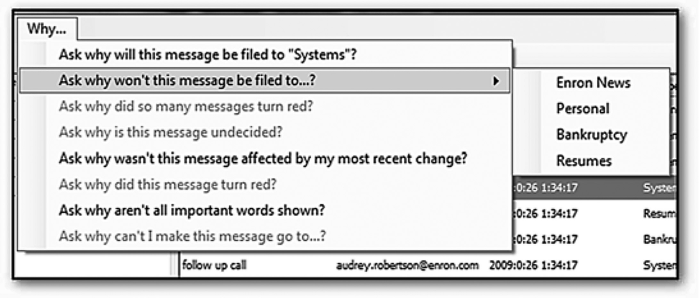

Why...
Ask why will this message be filed to "Systems"?
Ask why won't this message be filed to...?
Ask why did so many messages turn red?
Ask why is this message undecided?
Ask why wasn't this message affected by my most recent change?
Ask why did this message turn red?
Ask why aren't all important words shown?
Ask why can't I make this message go to...?
Enron News
Personal
Bankruptcy
Resumes
0:26 1:34:17
0:26 1:34:17
0:26 1:34:17
follow up call
audrey.robertson@enron.com
2009:0:26 1:34:17
Systen
Resum
Bankru
Systen

Fig. 1. The Why questions as presented in our prototype.

3.2. Design of Interactive Why Answers

In the original Whyline, answers to “why” questions pointed to relevant places in the source code, but in intelligent assistants, there is no source code that represents the assistant’s learned logic for end users to look at or modify. The interactive answer would have to represent the assistant’s logic by some device other than source code, while still allowing end users to edit it effectively.

Thus, we designed the interactive answers according to four principles.

—*Principle Representation*-1. Representations of both the assistant’s logic and execution state should be available to and manipulable by end-user debuggers.
—*Principle Representation*-2. Explanations of the assistant’s logic and execution state should not obscure or overly simplify the learning system’s actual reasoning.
—*Principle ML*-1. The intelligent assistant should be explainable to end users.
—*Principle ML*-2. The intelligent assistant should heed end users’ corrections.

Principle Representation-1 was inspired by the information content that traditional debugging systems provide. Such systems provide a representation of the logic (source code) that the programmer can edit. They also provide ways to inspect concrete data about program execution states. For example, debuggers provide access to the values of variables and the stack.

Principle Representation-2 boils down to being fully truthful with the user about the assistant’s logic. Machine learning often uses complex statistical models, so there is a temptation to hide complex details from the end user in an effort to make the system easier to understand. For example, naive Bayes considers both the presence and absence of features when classifying items; an explanation that only contains information for features present in an item may be easier to understand, but it obscures part of the learning system’s logic. Recent research [Tullio et al. 2007] has showed that end users could understand the logic that machine learning systems use to make their decisions, at least in a rule-based system. Further, obscuring this logic would break the debugging “principle of immediacy” [Ungar et al. 1997] by creating an artificial division between the logic that end users can view and interact with, and the underlying logic actually responsible for the assistant’s output. Thus, abstracting away portions of the assistant’s logic may hamper, rather than help, the user’s debugging efforts.

Principle ML-1 is almost a corollary to Principle Representation-2. We wanted to support underlying learning algorithms more complex than the trivially explainable decision trees, but not algorithms so complex that supporting Principle Representation-2 would become impossible. Regarding Principle ML-2, we also require that the underlying learning algorithm must be responsive to the user’s interactive manipulations to the logic representation. We expand upon both of these points in the next section.

3.3. The Design Principles and the Underlying Machine Learning Engine

With these four design principles in mind, we chose the naive Bayes [Maron 1961; Russell and Norvig 2003] algorithm for our prototype. Naive Bayes is a widely used probabilistic algorithm for text classification. It is more complex than ordinary decision trees, yet potentially explainable to end users (Principle ML-1).

The machine learning features in our email prediction application were the words contained in actual email messages. Given our choice of naive Bayes as the learning algorithm, these features and their associated predictive value for a classification are a complete representation of the assistant's logic. The *execution state* of the assistant is therefore the current set of predictions resulting from this logic, including the assistant's degree of certainty in each prediction. This application of Principle Representation-1 thus results in user-manipulable logic (words and their associated importance), changes to which affect the assistant's execution state (its predictions and their associated confidence).

Although our previous work [Stumpf et al. 2007] found that rule-based explanations were the most easily understood type of explanation, Principle Representation-2 stipulates that the explanation should accurately reflect the logic of the assistant. Thus, given our use of naive Bayes, we chose keyword-based explanations consisting of words and their weights. We refer to the weight of a word as the probability of the word being present, given the item's predicted class label. To implement Principle Representation-2, the visualization is a truthful reflection of the logic of naive Bayes, and hence displays the union of all words in all email messages. We allowed users to adjust the weights of any word. Because naive Bayes classifiers use class distributions to inform their predictions, the distribution of messages across folders is also displayed to the end user. Hence, the visualization and manipulations allow the user to directly change the logic the assistant should follow, based on an accurate representation of how the logic works.

Principle ML-2 says that corrections end users give the system should be heeded. We used a naive Bayes algorithm,[1] but modified the implementation such that corrections can be integrated in a straightforward manner and, when the user modifies the weight of a word, the classifier can set the new value to be close to the value specified by the user.

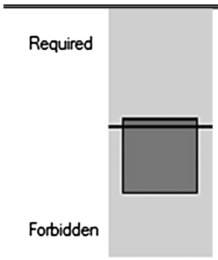
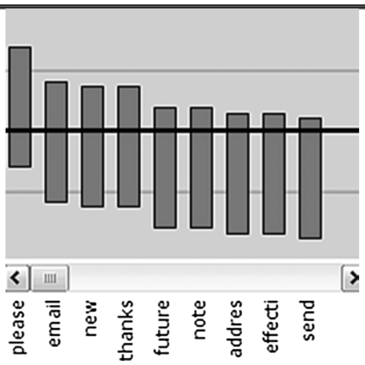
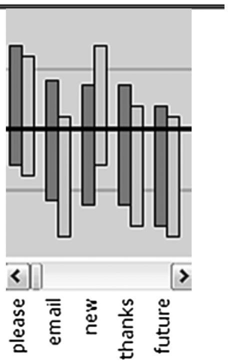
Being responsive to user corrections can be problematic because changing the weight of one out of thousands of available features (e.g., changing the weight on one word out of the approximately 6,000 different words contained in our email messages) could have little noticeable impact on classification. To make the classifier more sensitive to user feedback in the interactive setting, we treat the user-specified folder assignment as a new training data point for the classifier.

3.4. End-User Debugging with Answers

From these design choices, the answers took the following form. Textual answers are provided for all questions, giving high-level explanations about the logic of the assistant and general suggestions about how to debug it, but three questions are also answered through detailed visualizations of the assistant's logic: these answers formed the debugging mechanism for the end user.

[1] We had considered using user cotraining [Stumpf et al. 2009] as the underlying algorithm. When user co-training receives new information from the user about a feature's value to a folder, this technique assigns a new weight, based on a *combination* of the user-assigned value and the classifier's internal weight—which could potentially be quite different from the user-assigned value. In our previous work with user cotraining [Stumpf et al. 2008], we observed that this behavior was frustrating to users because it made the algorithm appear to "disobey" the user's corrections.

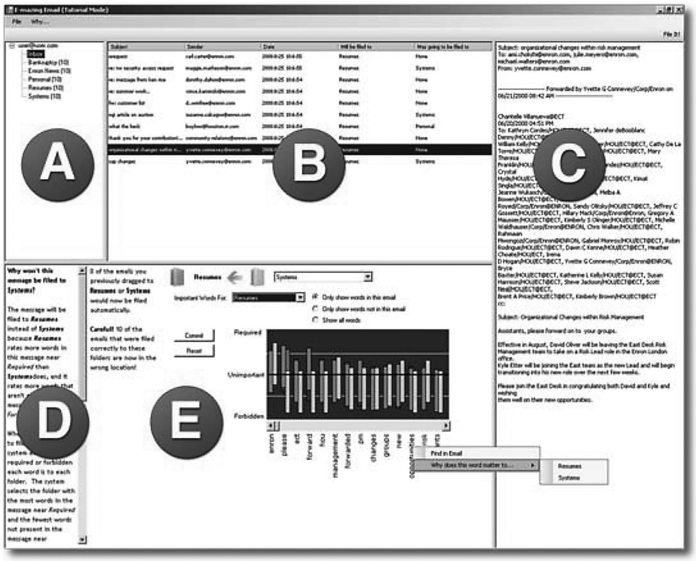
Table III. Visual Explanations for Three Why Questions

Why does <word> matter to <folder>?	**Why will this message be filed to <folder>?**	**Why won't this message be filed to <folder>?**
	 please email new thanks future note addres effecti send	

These answer visualizations are shown in Table III. Similar to ExplainD [Poulin et al. 2006], bars represent the importance of each feature to a given classification. However, rather than using bar area to indicate a feature's contribution toward the learning system's prediction, we indicated importance via bar location. By changing a bar's location, the feature's weight can be adjusted in real-time. The choice of bars, as opposed to points or some other representation, was made because their large target size makes mouse-based interaction and visual discrimination easy. To debug the assistant's faulty logic, users can change these answers by manipulating the bars (which operate like sliders or faders), with the weight of evidence represented by each bar being the midpoint of the bar. Because we were interested in exploring how end users debug an assistant's logic, as opposed to how they provide it with training data, the learning system did not train itself on messages that users' classified manually.

Providing the necessary interactive aspect for these bar visualizations required support from the underlying machine learning algorithm. Before explaining how these visualizations were implemented, we formally define the following properties of the naive Bayes algorithm. An email message is represented as a "bag of words", that is, a Boolean vector $W = (W_1, \ldots, W_m)$ in which W_i takes the value *true* if the ith word of a vocabulary of m words is present in the email message, and *false* otherwise. The vocabulary in our experiment consists of the union of the words from the following parts of all the emails: the message body, the subject line, and email addresses in the "To", "From," and "CC" parts of the message header. Stop words, which are common words with little predictive value (such as "a" and "the") were not included in the vocabulary.

3.4.1. Answering: "Why will this message be filed in <Folder>?" and "Why does <Word> matter to <Folder>?". In previous work [Stumpf et al. 2007], we observed that end users understood how the presence of keywords influenced a message's classification, but they struggled with the concept of how the absence of words influenced the same classification. We addressed this difficulty in a novel way by showing the weight associated with each word in the vocabulary via a bar positioned between the two extremes of *Required* and *Forbidden*, shown in the leftmost images of Table III. For folder f, the weight of a word is the probability $P(W_i = true \mid F = f)$ where W_i is the random variable for the ith word and F is the random variable for the folder. The closer the bar is to *Required*, the more important the presence of the word is to the prediction. If the top of the bar is at its highest position, then $P(W = true \mid F = f) = 1.0$. If the top of the bar is on the black line



Why won't this message be filed to Systems?

The message will be filed to Resumes instead of Systems because Resumes rates more words in this message near Required than Systems does, and it rates more words that aren't [illegible]

[illegible] required or forbidden each word is to each folder. The system selects the folder with the most words in the message near Required and the fewest words not present in the message near

0 of the emails you previously dragged to Resumes or Systems would now be filed automatically.

Careful! 10 of the emails that were filed correctly to these folders are now in the wrong location!

● Only show words in this email
○ Only show words not in this email
○ Show all words

Subject: organizational changes within risk management
To: ami.chokshi@enron.com, julie.meyers@enron.com, michael.walters@enron.com
From: yvette.connevey@enron.com

---------------------- Forwarded by Yvette G Connevey/Corp/Enron on 06/21/2000 08:42 AM ---------------------------

Subject: Organizational Changes within Risk Management

Assistants, please forward on to your groups.

Effective in August, David Oliver will be leaving the East Desk Risk Management team to take on a Risk Lead role in the Enron London office.
Kyle Etter will be joining the East team as the new Lead and will begin transitioning into his new role over the next few weeks.

Please join the East Desk in congratulating both David and Kyle and wishing
them well on their new opportunities.

Fig. 2. Screenshot of the prototype.

in the middle, then $P(W = true \mid F = f\,) = 0.0$. Since $P(W_i = false \mid F = f) = 1.0\text{-}P(W_i = true \mid F = f\,)$, the position of the bottom of the bar can also be interpreted as the probability of the absence of the word. If the bottom of the bar is at its lowest position, (i.e. closest to *Forbidden*), then $P(W = true \mid F = f\,) = 0.0$. In reality, although the end user is able to set the bars to its highest or lowest positions, the probability $P(W \mid F = f)$ is never set to the extreme values of 1.0 or 0.0, due to the naive Bayes algorithm's use of Dirichlet priors to smooth the probabilities. As a result, the probabilities will be an epsilon different from the extremes and the bar will be positioned slightly below/above the respective Required/Forbidden level in the visualization.

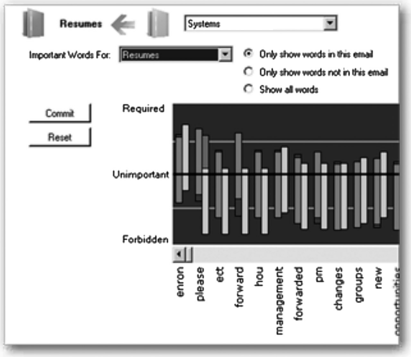
Aside from the probability $P(W \mid F)$, the naive Bayes classifier also uses the class probability $P(F)$. Although we show the number of messages in each folder, we do not display $P(F)$ as an interactive bar graph because the training set initially consisted of an equal number of emails in each folder, and these numbers were displayed beside each folder name.

3.4.2. Answering: "Why won't this message be filed in <Folder>?". The answer to this question is in the form of a dual-bar view that allows the user to compare and contrast the importance of words between the two folders, shown in the right image of Table III. The bars show the respective weights for the currently predicted folder f, and the other folder f' the user indicated, where the positions of the bars correspond to $P(W_i = true \mid F = f\,)$ and $P(W_i = true \mid F = f')$, respectively.

We can illustrate the degree that an email "belongs" to either folder f or f' using the arrow, as shown at the top of Figure 3. The angle of the arrow between the folders is based on the difference between $P(F = f' \mid W_1, \ldots, W_m)$ and $P(\,F = f \mid W_1, \ldots, W_m)$. This also serves to reveal the execution state to the end user. Since the bars allow weights associated with the two folders f and f' to be manipulated, changes to individual words by the end user that result in $P(F = f' \mid W_1, \ldots, W_m) > P(F = f \mid W_1, \ldots, W_m)$ will be shown by the arrow moving to point to folder f' instead.

3.5. Debugging Scenario Using the Prototype

Figure 2 shows how the debugging supports we have described are brought together in our email prototype. The user's view consists of familiar email client elements: a



Resumes ← Systems

Important Words For: Resumes

● Only show words in this email
○ Only show words not in this email
○ Show all words

Fig. 3. Close-up of the visualization/debugging features. The user has just decreased the importance of “please” to Systems by dragging the blue (light) bar downward, but the system still thinks the message belongs in *Resumes*.

folder list (A), a list of headers in the current folder (B), and the current message (C). In addition, it includes debugging panes that hold textual answers (D) and interactive visualizations (E).

If at some point the user wants to know why the assistant is behaving in a particular way, she can ask Why questions either through the global menu bar or a context-sensitive menu and receive an answer which supports debugging. Consider the following scenario: A user notices that her intelligent assistant thinks a message about *Systems* belongs in the *Resumes* folder. In Figure 3, this user has just asked why the message is not filed under *Systems*. The keyword bar graph shows the system’s opinion of the importance of each word to the *Resumes* folder (dark pink), which is the current folder for this message, versus importance to the *Systems* folder (light blue). While reviewing this explanation, the user quickly identifies a problem—her assistant thinks the word “please” is equally important to both folders. The user disagrees; she wants to tell her intelligent assistant that she rarely sees the word “please” in messages from the *Systems* group. Thus, she drags the light blue bar lower (second from left); how much lower depends on her assessment of how important “please” should be to the *Systems* folder. The dark blue bar indicates the original importance of “please,” allowing the user to see her change and its magnitude.

User changes to each bar graph entry cause the system to immediately recalculate its predictions for every message in the inbox, allowing users to instantly see the impact of their manipulations. These changed folder predictions are shown through a change in the direction of the arrow between the two folders for the currently viewed message. They are also listed textually next to each message header in the inbox, highlighting headers whose predictions have changed with a red background. For every manipulation, the user immediately sees how both the assistant’s logic (in terms of the importance of words), and the execution state (e.g., the resulting program’s predictions), have changed.

4. ADDRESSING DEBUGGING BARRIERS AND INFORMATION NEEDS USING THE WHY-ORIENTED APPROACH

To investigate obstacles end users might encounter while correcting an intelligent assistant using our new approach, we conducted an exploratory study using the prototype we have just described. Our purpose was not a summative evaluation of the prototype, but rather to investigate three questions to understand how our approach could be extended to further support end-user debugging: when and where the end-user debuggers encounter problem areas (barriers); what could help users when they encounter these problem areas (information needs); and how machine learning algorithms could help address the barriers and information needs. Further, since researchers have recently found evidence of gender differences in debugging, we investigated gender as an aspect to highlight design considerations that could differently impact subsets of end users.

4.1. Exploratory Study Design

4.1.1. Study Design and Participants. The study used a dialogue-based think-aloud design, in which pairs of users talked to each other while collaborating on a task. The pair design was to encourage "thinking aloud" by leveraging the social norms that encourage people to voice their reasoning and justifications for actions when working with a partner.

Six pairs of female and five pairs of male students participated. The pairs were evenly distributed by gender across GPAs, years in university, and email experience. All twenty-two participants were required to have previous email experience and no computer science background. In order to eliminate a lack of familiarity with each other as a potential difficulty, pairs were required to sign up together. Pairs also had to be the same gender, so that we could clearly identify any gender differences that might arise.

We ran the study one pair at a time. Each session started with the participants completing a questionnaire that asked for background information and gathered standard presession self-efficacy data [Compeau and Higgins 1995]. We then familiarized the pair with the software and examples of classification through a 20-minute hands-on tutorial. An instructor read a scripted tutorial, teaching participants about the filing and classification abilities of the prototype, how to ask the prototype questions, and giving an overview of the prototype's answers and how to change those answers. The same instructor presented the tutorial to each participant pair.

For the study's main task, participants were asked to imagine they were co-workers in a corporate department at Enron. Their department included a shared email account to provide easy access to communications that affected all of them. The premise was that new email software, featuring the ability to learn from the users and automatically classify messages into a set of existing folders, had recently been installed; their supervisor then asked them to get messages from the inbox into the appropriate folders as quickly as possible, doing so in a way that would help improve later classifications. Because the prototype's classifier did not train on (and thus, did not learn from) messages that the participants manually dragged to folders, the task encouraged participants to debug the system via the interactive explanations.

We used the publicly available Enron email data set in our study. To simulate a shared mailbox, we combined messages that three users (farmer-d, kaminski-v, and lokay-m) had originally filed into five folders (Bankruptcy, Enron News, Personal, Resumes, and Systems). At the start of the session, each folder held 20 messages that were used to initially train the classifier. Participants were given five minutes prior to the main task to familiarize themselves with the existing categorization scheme (with the 20 training messages in each folder available as examples), so that they would have an idea of how new messages could be filed. They were told that there was no "correct" classification; they should file the messages in a way that made sense to them. The inbox contained 50 additional messages for the participants to file. The amount of training data was small so as to simulate real-world situations, in which users have not invested the time to label hundreds of training examples.

Each pair worked on the main task for 40 minutes, with participants switching control of the mouse half way through the session. We used Morae software to capture video and audio of each user session, synchronized with their screen activity. Our prototype also logged each user action. After completing the main task, participants individually filled out a questionnaire, gathering feedback about the prototype and their postsession self-efficacy.

4.1.2. Analysis Methodology. To analyze the participants' verbal protocol during our study, we developed two code sets to capture barriers and debugging activities. Ko et al. identified six types of learning barriers experienced by novice programmers using a new programming environment [Ko et al. 2004]; these barriers informed our investigation because our participants, like theirs, were problem-solving about how to make programs work correctly and were inexperienced with the facilities provided for debugging. Because our debugging environment is substantially different from the traditional textual programming environments studied in Ko et al. [2004], we adjusted the definitions of Ko's barriers to map the high-level problems each barrier describes to the problems participants faced while debugging with our prototype (examples are from actual participant transcripts).

—Ko's *Design barriers* are situations in which the user does not know exactly what he or she wants the computer to do. When debugging an intelligent assistant, this means the user doesn't have a clear idea of how to go about fixing the assistant. For example, "Can we just click *File It*?"
—Ko's *Selection barriers* occur when the user knows what he or she wants the computer to do, but does not know which programming tools to select to accomplish the goal. Our why-oriented debugging tool has the ability to adjust the assistant's logic via feature modification, so we mapped selection barriers to end-user difficulties in selecting which feature to adjust. For example, "What kind of words should tell the computer to [file this] to *Systems*?"
—Ko's *Coordination barriers* occur when the user knows the programming tools he or she wants to use, but does not know how to make them work together. With our debugging tool's affordance to modify the features the assistant uses to classify items, such modifications may alter multiple classifications. Thus, coordination barriers in this domain reflect the difficulty of coordinating how independent changes to the assistant's logic have resulted in the current set of predictions. For example, "OK, I don't know why it [changed classification]."
—Ko's *Use barriers* are situations where the end user knows which programming tools he or she wants to use, but does not know how to use them properly. The UI affordance in our approach for feature modification is to drag a slider up (to increase the importance of a feature's presence) or down (to increase the importance of a feature's absence). Trouble deciding in which direction or to what extent to adjust this slider is our version of use barriers. For example, "So is [this word] 'unimportant'?"
—Ko's *Understanding barriers* occur when end users thought they knew what to do, but the results of their actions were surprising. This barrier mapped directly to situations in which users were surprised by the assistant's feedback. For example, "Why is 'web' more forbidden for [the] *Systems* [folder]?"

We did not use Ko et al.'s sixth barrier (searching for external validation), because problem-solving in our study was based on facts internal to our environment. Regarding debugging activities, previous research [Davies 1996; Ko 2008] identified six common actions in fixing bugs in programming environments. We applied the two not involving data structuring or writing new source code, and also introduced a fault detection code.

—*Fault detection* denotes participants detecting an incorrect prediction by the system. For example, "It's going to [the] *Systems* folder; we do not want *Systems*."
—*Diagnosing* occurs when participants believe they have diagnosed the specific *cause* of a detected fault. For example, "Well, [the importance of] 'email' needs to be higher."

Table IV. Example of our Code Set Applied to a Participant Transcript

Minute	Participant	Utterance	Code
21	P1	Why everything is going to Personal?	Coordination
21	P2	Can we just drag or do we have to make system learn?	Design
21	P1	We'd have to drag every one.	Hypothesizing
23	P2	I think we should file email via chart to adjust more easily.	Hypothesizing
24	P2	Should we try 'chairman'? [receive message from gtalk asking to switch places] [clicks on different message: 'eol outage Saturday january 20th']	Selection
24	P2	Then you want to adjust something.	Hypothesizing
24	P1	See 'eol' should be Systems not Personal.	Fault Detection

—*Hypothesizing* represents times when participants hypothesized a general, rather than a specific, solution to detected faults. For example, "Let's move something else, and then maybe it'll move [the message] to *Systems*."

We applied the codes to "turns". A turn consists of sentences spoken by a participant until his or her partner next spoke. Speech by one participant that contained a significant pause was segmented into two turns. If the same barrier spanned multiple turns (for example, if one person was interrupted by the other), only the first occurrence of the barrier was coded. Only one code could be applied per turn.

Working iteratively, two researchers independently coded a randomly selected five-minute section of a transcript. We determined similarity of coding by calculating the Jaccard index, dividing the size of the intersection of codes by the size of the union for each turn, and then averaging over all turns. Disagreements led to refinements in coding rules that were tested in the next coding iteration. Agreement eventually reached 82% for a five-minute transcript section, followed by an agreement of 81% for a complete 40-minute transcript. Given this acceptable level of reliability, the two researchers divided the coding of the remaining transcripts between themselves. A three-minute section of a coded transcript is included in Table IV to illustrate the application of our final code set.

4.2. Results: Debugging Barriers (*When* and *Where*)

4.2.1. Barriers Encountered. Our participants' attempts to debug ran into numerous barriers, encountering an average of 29 barriers during the 40-minute study (with a range from 7 to 66). The number of barriers encountered was about double the number of messages participants filed (mean = 16.1, SD = 8.3). Time did not matter: these barriers were equally likely to be encountered at the beginning and end of the study. Everyone hit barriers, but confidence mattered: participant self-efficacy (a specific form of self-confidence [Bandura 1977]) was significantly predictive of the number of barriers they encountered, normalized for the number of utterances each participant made (linear regression, $F(1,9) = 7.11$, $R^2 = 0.44$, $beta = -0.005$, $p = .026$).

As Figure 4 shows, the most frequent barriers our participants encountered were *Selection* barriers (40% of all barriers), that is, selecting the right words or messages to modify to fix the assistant. In Ko et al's work [2004] on traditional programming environments, *Selection* barriers did not play a large role, but they did play a large role in our domain. This participant, for example, had trouble determining which features are important for the *Systems* classification.

> P712: "Then 'news'? Well, they like team players. Contributions? That would be more that you'd use for News than Systems."

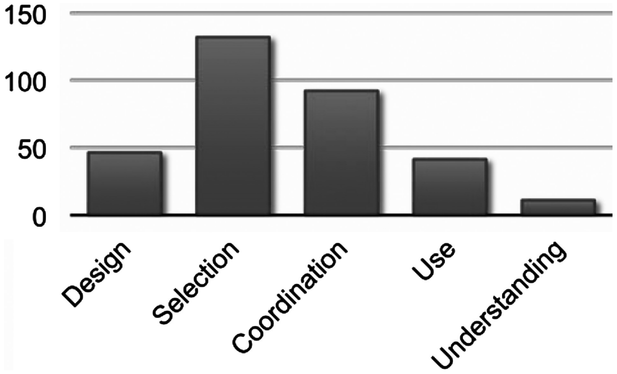


Fig. 4. Sum of barriers encountered in all transcripts.

Coordination barriers also occured frequently (28% of all barriers). Participants often wondered how the corrections they were about to make would change the system's other predictions, as well as how to coordinate the systems response (or failure to respond) when they adjusted the assistant's logic.

P732: "Resume? [user finds word, makes 'resume' required] Why didn't it change it?"

Further evidence that *Selection* and *Coordination* barriers were especially problematic comes from the participants' responses to a questionnaire where 16 of 22 respondents (72%) mentioned difficulty in determining which words were important when fixing misclassified mail.

Our results highlight that the kinds of barriers encountered while debugging intelligent assistants may be rather different than those seen in traditional programming environments, and hence require their own specialized support. Ko's top two barriers in the more traditional programming environment were *Use* and *Understanding* [Ko et al. 2004]; but these two categories were fairly low for our participants (Use: 12%, Understanding: 3%); instead, *Selection* and *Coordination* barriers were our participants' top two obstacles. The differences among the *Use, Understanding,* and *Selection* barriers that our participants encountered and those faced by Ko's participants may be related to differences in the logical structure of traditional source code versus sets of features and predictions. In traditional source code, a user must select from among only a small number of constructs, each with different purposes and semantics, providing sizeable territory for use or understanding questions. In contrast, our Why-oriented approach had only one simple construct (a feature bar), which participants had no trouble using or understanding—but the participants had to select from a very large number of instances of that construct, providing plenty of potential for selection questions.

Regarding *Coordination* barriers, the statistical logic of machine learning systems may have further contributed to confusion when multiple predictions changed as a result of a single feature modification. Even if participants *did* expect such wide-ranging changes, it was sometimes difficult to coordinate which and how many predictions changed in response to each user modification, making it hard to judge whether the results are acceptable.

While *Design* and *Use* barriers should not be neglected, the predominance of *Selection* and *Coordination* barriers in this domain suggests that end users may have less trouble deciding on a strategy for *how* to give feedback (*Design* and *Use*), than on *where* to give feedback (*Selection* and *Coordination*).

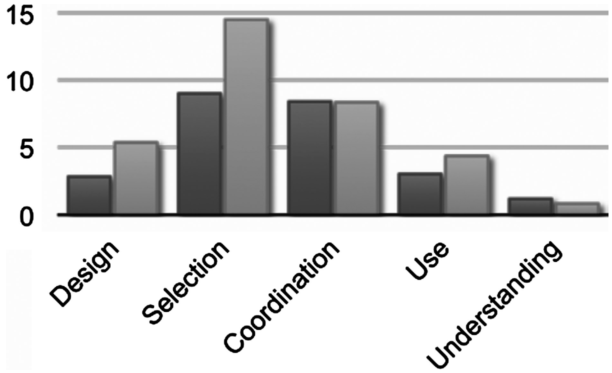


Fig. 5. Average number of barriers per session encountered by males (dark blue bars) and females (light pink bars).

4.2.2. Gender Differences in Barrier Encounters. Are barriers different for male and female end-user debuggers, and will overcoming them require different kinds of support? Interestingly, males and females did not appear to experience the same number of barriers: females encountered an average of 33.3 per session, versus the male average of 24.4 per session. This difference was despite the fact that males talked more (and thus had more opportunities to verbalize barriers) than females, averaging 354 turns per session, compared to 288 for females.

Figure 5 shows the average barrier count per session (the same differences are also present in comparing the average barrier counts per turn). Females experienced more barriers in almost every category; the only exceptions were *Coordination* and *Understanding*. *Selection* barriers, the most common barrier type, had a large difference: females averaged 14 per session, about 1.5 times as many as the male average of nine. This difference was statistically significant despite the small size of our sample population (Wilcoxon Rank-Sum Test: $Z = 2.1044$, $p < 0.05$). *Design* barriers, too, exhibited a strong contrast, with females averaging 5.3 per session versus males averaging 2.8. The differences in both the total number of barriers and *Design* barriers encountered were not statistically significant; but this may be a result of our small sample size (totaling six female and five male pairs). A statistically oriented experiment with a larger sample is needed to provide more conclusive evidence.

One reason for the apparent differences may be that females expected more problems due to lower self-efficacy (a form of self-confidence specific to the expectation of succeeding at an upcoming task [Bandura 1977]). As mentioned earlier, there was a significant, inverse relationship between self-efficacy and the number of barriers a participant encountered. Females came into the study with lower self-efficacy (measured via a self-efficacy questionnaire [Compeau and Higgins 1995]) than males, scoring an average of 38 out of a possible 50, compared to 42 for males (Wilcoxon Rank-Sum Test: $Z = -2.64$, $p < .01$). This is consistent with similar self-efficacy differences for end users engaging in other complex computer tasks [Beckwith et al. 2005, Grigoreanu et al. 2008; Subrahmaniyan et al. 2008]. As we show in the next section, our results about differences in barriers are consistent with prior research in another aspect as well: these prior works showed gender differences in both features used and the strategies end users employed to find and fix errors in spreadsheets.

Another possible cause for the observed disparities may be gender differences in information processing. For example, work on the selectivity theory of information processing [Meyers-Levy 1989] has shown a number of differences in how males and females process information. According to this theory, females are more likely to work with information comprehensively, whereas males are likely to pursue information more selectively. The following quotes illustrate the tendency our female pairs showed toward examining several words from a message before moving on, versus the males' propensity for advancing to the next message as quickly as possible.

Female Pair

P1131: "So that [word is] really important. And then, um, probably 'updates' would be important. And then, um... [the word] 'virus'?"
P1132: "Yeah. And then, uh, [the word] 'login'."

Male Pair

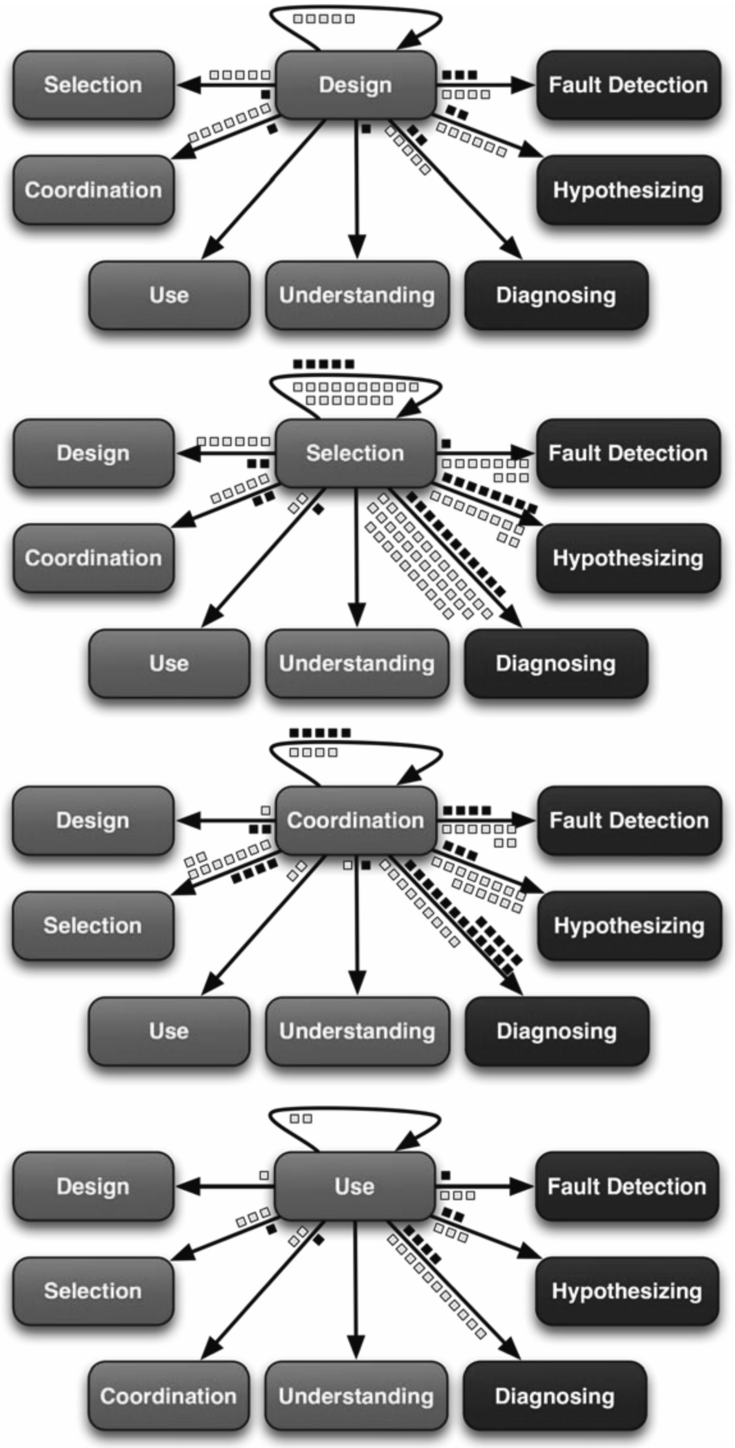
P1211: "Its [classification is] correct. It's learned something, eh."
P1212: "Um hmm."
P1211: "Lets go to the next message."

The selectivity theory is also consistent with our frequency data: females worked with a larger set of words than males did (106 unique words for females vs. 62 for males), perhaps to perfect the algorithm's performance. Males, conversely, may have been more inclined to move on to the next message as soon as they obtained the desired effect. Supporting this hypothesis is the average number of messages each group filed—male pairs averaged 19.8 messages filed in each 40-minute session, while female pairs averaged 13.0. This suggests that in order to successfully support a wide range of end users, debugging features should be designed so that both comprehensive and selective strategies can lead to success.

4.2.3. Barrier Transitions and Gender Differences. When a participant encountered a barrier, what happened next? Were there different patterns of what male and female participants did after encountering a barrier? Understanding how barriers were encountered and resolved (or not) can provide insight into their severity and impact, and highlight when additional support may be most necessary.

To answer these questions, we investigated the sequence of barriers encountered and looked for differences between male and female participants. The barriers and debugging activities coded in the participants' verbalizations are simply states between which they can transition. To calculate the probability of each state (barrier or activity) following an initial barrier, we divided the number of occurrences of a particular subsequent state by the total number of states that followed the initial barrier. For example, if *Selection* followed *Design* once and *Diagnosing* followed *Design* twice, then the probability of *Selection* following *Design* was computed as $1/(1 + 2) = 0.33$, or 33%, and the probability of *Diagnosing* following *Design* was computed as $2/(1 + 2) = 0.66$, or 66%. We use these probabilities for clarity only; our graphs (Figures 6, 7, 8, and 9) show the exact number of instances for completeness.

Initially, we hypothesized that there would be a commonly followed path through the barriers, which could allow us to support debugging by providing the right information at just the right time. For example, perhaps most users would flip between *Design* and *Understanding* barriers at the beginning of the task. Once those early hurdles were overcome, they might cycle through *Selection*, *Coordination*, and *Use* barriers until they understood the assistant's logic, after which the barriers would be encountered with considerably less frequency. When examining our participants' data, however, we saw no discernible tendency for users to encounter barriers in specific orders or at consistent times during their debugging session. Nor did barrier frequency change in any significant way over time; users were just as likely to encounter *Design* barriers in the first five minutes of the study as they were in the final five minutes. This suggests



Figs. 6, 7, 8, and 9 (from top to bottom). Number of transitions from barriers to other barriers or debugging activities. Light squares indicate one instance by a female pair; dark squares indicate one instance by a male pair. Barrier nodes are colored light purple; debugging activity nodes are dark blue.

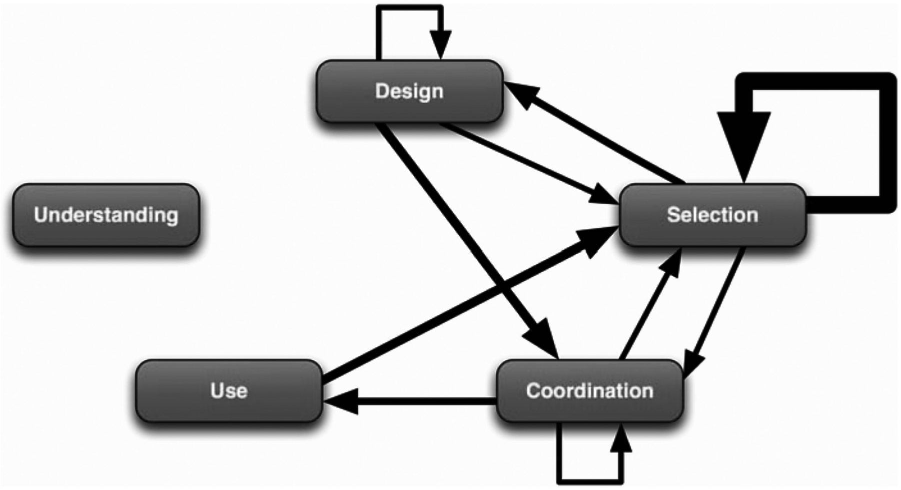


Fig. 10. An example participant pair's path through the debugging barriers. The width of the arrows indicate the percentage of transitions: thinnest = 6%, thickest = 20%. Transitions accounting for 5% or less of total are not shown.

that an approach aiming to support end-user debuggers in overcoming barriers must support finer-grained patterns of barriers as they are encountered, instead of basing its explanations on an expected common path.

Both male and female end users exhibited finer-grained patterns for many of the individual barriers. Although no single barrier stood out as a frequent transition from *Design* (Figure 6), when we examined gender differences, we found that males reacted to *Design* barriers with some form of debugging activity on average 70% of the time, versus 46% for females. Thus, it appears that *Design* barriers proved less of a problem to males than females, since they were able to move on to a debugging activity instead of encountering another barrier. Since *Design* barriers relate to a user lacking a debugging strategy, an implication of this gender difference is that our approach's explanations for overcoming these barriers should be certain to include strategies other researchers have identified as frequently employed by female end-user debuggers [Subrahmaniyan et al. 2008].

Figure 7 shows that the most common action following a *Selection* barrier was the debugging activity *Diagnosing,* which occurred after 40% of *Selection* barriers. The next most prevalent code was a second *Selection* barrier (19%), suggesting that *Selection* barriers were either quickly overcome, leading to *Diagnosing*, or they cascaded, stalling participants' debugging progress. This implies that once users had trouble deciding where to give feedback, they became less and less able to do so. Figure 10 illustrates the problem by graphing all of the barrier transitions for one of our participant pairs (P701 and P702). The high number of incoming edges to the *Selection* box was typical, as is the loop from *Selection* back to itself. We already discussed the high number of *Selection* barriers overall, but Figure 7 points to those barriers specifically stalling participants; an initial *Selection* barrier was often followed by a second. Further, our participants were paid for their time and worked in teams; we hypothesize that in real-world applications, two or three sequential *Selection* barriers (which prevented participants from making visible progress) may be enough to dissuade individual end users from investing time and effort in improving their intelligent assistants. This suggests the need for our approach to point out the words or features that would be most likely to change the program's behavior; we discuss how this might be done in

Section 4.4.1. These participants, for example, could have benefited from the following of help.

P732: “And what about ‘interview’? Oh, we just did that, so no. ‘Working’, maybe?” [finds word]
P731: “Well, no because ‘working’ could be used for anything really.”
P732: “True.”
P731: “‘Work’, no.”
P732: “What about... [scrolls left] ‘scheduling’. No, that could be News.”
P731: “That could be News, too.”
P732: “What about ‘scientist’?”
P731: “That could be Personal.”

What about those users who did *not* stall on *Selection* barriers? Males had a higher tendency for *Hypothesizing* following a *Selection* barrier than females, 26% to 11%. Recall that *Hypothesizing* was coded when the pair discussed a possible fix but didn’t include a specific word, whereas *Diagnosing* indicates that the pair specified the word they intended to modify. Thus, males were more likely to follow a *Selection* barrier with a general solution, while females tended to first agree on a word to alter. Why this difference? We know our female participants came into the study with lower self-efficacy than our male participants, and prior research [Beckwith 2007] has revealed female end users to be more risk averse, in general, than male end users. Both low self-efficacy and risk-aversion may have been alleviated by the pair coming to agreement about the best way to proceed; a participant’s self-efficacy could be boosted by discovering that her partner agrees with her idea, and this improved confidence may in turn lower the perceived risk of the proposed debugging fix. Our approach could use the same solution proposed to help users overcome *Selection* barriers (directing end users toward words that will have the strongest effect on message reclassification) to help low self-efficacy users as well, by reducing the choice of words to modify down to a more manageable, less intimidating, subset.

Like *Selection* barriers, *Coordination* barriers often led to *Diagnosing* (30%) (Figure 8). Taken together with the other two debugging actions, *Fault Detection* (14%) and *Hypothesizing* (20%), this barrier was followed by a debugging action 65% of the time. Males, however, tended to follow *Coordination* barriers with more *Diagnosing* than females (47% vs. 18%, respectively), whereas females followed them with more *Hypothesizing* than males (29% vs. 8%). One interpretation of these results is that following the confusion regarding the impact of their changes, female participants were more likely to step back and attempt to coordinate how their changes will impact the entire system, whereas males tended to stay focused on a specific failure. This would be yet another indication of the comprehensive problem-solving strategy associated with females [Meyers-Levy 1989], providing further evidence for the need to support both comprehensive and noncomprehensive problem-solving strategies in end-user debugging environments.

Finally, *Use* barriers (Figure 9) were strongly tied with a transition to *Diagnosing* (44%); all other transitions were below 15%. It seems that when a *Use* barrier was encountered, our participants’ response was to adjust their specific solution, rather than move to a different problem or generalize a solution. This was equally true for males and females. It appears that participants can frequently overcome this barrier by themselves and only need additional help in a few instances.

Thus, the patterns of barrier transitions are consistent with the Selectivity Hypothesis information processing strategies discussed earlier, adding additional emphasis to the need for intelligent assistant debugging approaches to support both selective and compressive information processing strategies.

4.3. Results: Information Needs (*What* Could Help, and *When*)

4.3.1. Barrier Transitions and Gender Differences. What information would have helped our participants overcome these debugging barriers? To explore this, we compiled a list of every question participants asked during the course of the study, and two researchers worked together to group the questions based on the type of information being requested. This grouping resulted in the seven knowledge bases described in Table V. The information needs were nearly always in transcript segments that also contained a barrier (as coded in Section 4.1.2). Thus we were able to calculate how often they wanted each kind of information when encountering each barrier. Table V summarizes the results: each graph illustrates the percentage of barriers in which participants explicitly requested a specific type of information to overcome the barrier. Figure 11 shows the frequency of each category.

As Figure 11 illustrates, the most frequent information requested by end users was concrete advice about how to fix the machine's logic. Users wanted to know specifically *which* word weights they should be modifying to move a message into a specific folder; *how much* they should be adjusting the word weights; and previews of *what will happen* after adjusting the weights of particular words. This type of information alone represented nearly half of the total information requests among participants (42% of all requests), suggesting that even satisfying this information need alone may address a large number of user difficulties. Our participants most frequently discussed the value of this information after encountering either a *Selection* (71%) or *Use* (86%) barrier, suggesting that satisfying these information needs via answers to our Why questions could almost eliminate these two barriers.

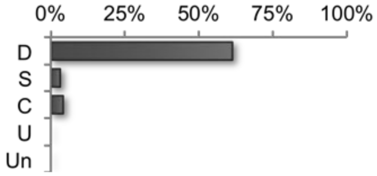
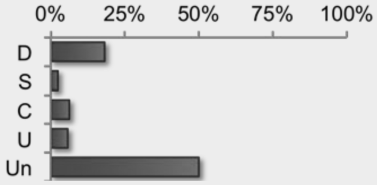
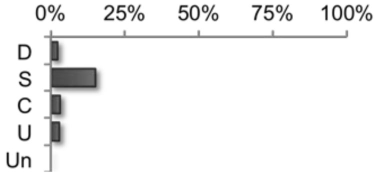
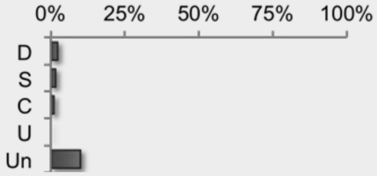
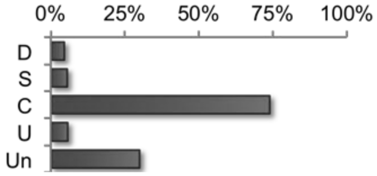
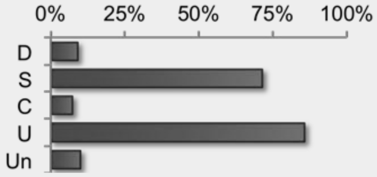
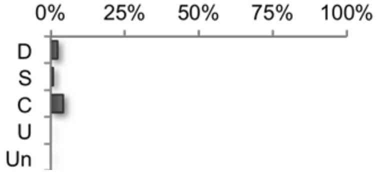
The second largest set of user-requested information related to the intelligent assistant's current logic. This includes user questions about why the intelligent assistant is behaving in a particular manner, such as "Why did this message turn red?" and "Why won't this message be filed to *Systems*?" Participants most frequently expressed a need for this type of information after encountering a *Coordination* barrier (74%). This was the primary set of questions we hoped to answer via our Why-oriented debugging approach. Note that, as with the information needs on how to fix the assistant's logic, users appeared to want concrete information–they mentioned specific messages and folders, rather than general explanations on how the assistant makes predictions. This desire for concrete solutions highlighted a nuance in our prototype: it coupled concrete visual explanations of word weights with generalized textual explanations of the assistant's general algorithm for classifying messages. Participants frequently commented that the generalized textual answers to their questions did not change with context and were unhelpful. This is an example.

> P1001: Ok go to the 'why' button. [clicks on the global why menu] Ask 'why not Systems'. [asks 'why won't this message be filed to..?'].
> P1002: It says the same thing about word importance.

Thus, phrasing the explanation in the context of the message that a user is currently working with may be a better way to present this particular type of information.

When participants encountered a *Design* barrier, they frequently believed they could overcome it if they possessed more details about debugging strategies (61% of *Design* barriers). These strategy questions fell into two categories: general, when the end user appeared to be searching for a different, better strategy (e.g., "Do we have to teach the system everything?"); and refinement, when the user had questions about particular aspects of their current strategy (e.g., "Should we file it now?"). Note that participants expressed more interest in information about debugging strategies than about the user interface features of our Why-oriented prototype. Prior research [Kissinger et al. 2006]

Table V. Participants' Information Needs when Encountering Barriers. (Graphs indicate the percentage of *Design* (D), *Selection* (S), *Coordination* (C), *Use* (U), and *Understanding* (Un) barriers in which participants explicitly mentioned needing the left column's information to overcome each barrier.)

Participants Asked (paraphrased quotes)...	When Would This Help?
Debugging Strategy • Do we have to teach the system everything? • Is there an easy way to get through this? • Can we do this quicker? • Should we adjust anything at all? • Should we file it now?	
Explanations of User Interface Features • Can we go back and fix bad classifications? • Where's the button that was supposed to show up? • Which color is related to which folder? (e.g., the pink and blue bar graph)	
Overview of Existing User Data • How many messages are going to each folder? • Where are similar message currently filed? • Is [some word] used in other messages or folders?	
ML Algorithm's Capabilities • What parts of the message are used in classification? • Can the system handle phrases? • Can we include context of keywords? • Does changing a word weight affect other folders?	
ML Program's Current Logic • Why did this message turn red? • Why are more messages being filed to [folder]? • Why didn't anything change after my last action? • Why won't this message be filed to [folder]?	
How to Fix the ML Program's Logic • How important should [word] be? • Which word should we modify? • Which words should be 'forbidden'? • What if we...?	
User Action History • What have we done? • Were our changes helpful? • Was one of our past actions wrong?	

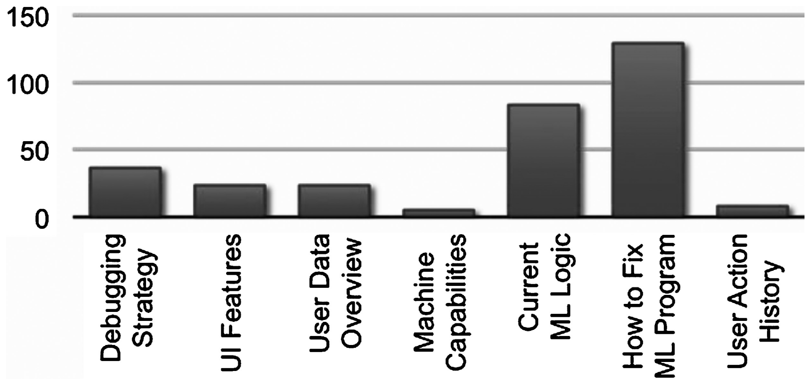


Fig. 11. Number of times a participant implied a particular knowledge base would help overcome their current debugging barrier.

regarding end users debugging spreadsheet formulas reported a similar difference, and posits that this is part of the phenomenon of "active users" [Carroll and Rosson 1987] who are primarily concerned with completing a task, rather than exploring "potentially interesting" user interface features.

Participants expressed an interest in overviews of the existing dataset they worked with, such as "How many messages are in each folder?" and "What other messages are [some word] used in?"—a need we also observed in a previous study [Stumpf et al. 2008]. Although these only accounted for 7% of observed information needs, a data overview was requested after 15% of *Selection* barriers, suggesting that explanations providing these details may help alleviate the most egregious barrier we observed in this domain. Further, every participant pair, save one, explicitly requested an overview of the data at least once. Part of the reason may have been due to unfamiliarity with the set of email messages our participants worked with, but even if they had such familiarity, such an overview could be useful to gauge the importance of words and the distribution of classifications quickly in large data sets.

The final two information needs, machine capabilities and user action history, were rare, accounting only 5% of all information requests, combined. Even though their occurrence was low, they may have fundamental effects; clearly, a flawed understanding of what the intelligent assistant is capable of parsing and using in its predictions would seriously impair one's ability to provide useful feedback to the assistant, and a clear indication of the actions a user has performed over time may help end users understand the long-term effects of their changes on the assistant's predictions. The idea of displaying the history of user actions is also consistent with the Whyline approach, since each user-made change to the assistant's logic can impact the classification of multiple messages, and similarly, a single change in classification may be the result of a collection of multiple user adjustments. Explanations of why such a change in classification occurred would be incomplete if they omitted the user modifications contributing to the reclassification.

4.3.2. Gender Differences in Information Needs. Since there were gender differences in the barriers encountered, we also investigated whether there were discernable gender differences in participant information needs that could help overcome these difficulties.

Although our data was sparse, female participants tended to have more questions about UI features than male participants (Figure 12). Previous research [Beckwith et al. 2005] has investigated gender differences in the adoption of debugging features for traditional programs. In that work, females were hesitant about adopting new features; they adopted them less frequently and later in the process than familiar

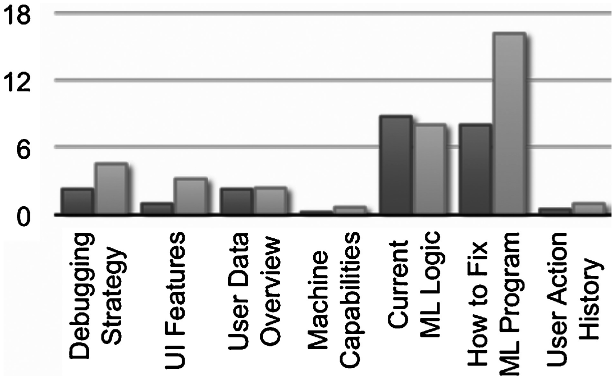


Fig. 12. Average number of information needs per session encountered by males (dark blue bars) and females (light pink bars).

features; and this had repercussions on new bugs introduced through their debugging activities. Our Why-oriented approach introduced a new set of features that are aimed at debugging intelligent assistants. Thus, providing specific support to help understand these features and entice females to use them may be warranted.

An interesting gender difference among our participants was their interest in strategic and tactical information. Figure 12 shows that information about *Debugging Strategy* and *How to Fix the ML Program* was requested twice as much by females as males. Information regarding *How to Fix the ML Program* is particularly applicable for overcoming *Selection* barriers (Table V), which females encountered almost twice as frequently as males. Moreover, females tended to encounter further *Selection* barriers after the initial *Selection* barrier; providing this information could reduce the risk of females becoming stuck in these *Selection* barrier loops. Furthermore, providing *How to Fix the ML Program* information could help females overcome *Design* barriers and move on to debugging activities (e.g., diagnosing, hypothesizing, or fault detection).

Recent work has explored the possibility of using textual and video explanations to present end users with debugging strategies [Subrahmaniyan et al. 2007]. Such techniques may be especially applicable in this domain because of low user self-efficacy regarding ability to debug an intelligent assistant, and to the inverse relationship we observed between participant self-efficacy and the number of barriers participants encountered. According to self-efficacy theory, watching similar people succeed at the same type of task not only helps to guide users toward successful strategies, but may also increase their confidence in being capable of completing the task themselves [Bandura 1977].

In contrast, we did not see gender differences in understanding how the intelligent assistant should operate. In our study, female participants requested an overview of existing user data and the machine's logic and capabilities at about the same frequency as males, suggesting no gender difference in reasoning about how the classifications should be made by the participants themselves.

In summary, it seems that any gender differences in information needs were more strongly tied to needing explanations about how to strategically make changes than to the specifics of the assistant's logic. Fulfilling these targeted information needs may help to remove the differences in barriers (particularly *Design* and *Selection*) that affected females more than males.

4.4. How Machine Learning Algorithms Can Better Help Users (How)

If the user and her machine are to be partners, each has a role in helping the other cope with difficulties. Our Why-oriented approach allows users to help improve the machine's logic; but how can the machine further help users with their barriers and information needs? We focus particularly on *Selection* and *Coordination* barriers, as these accounted for over two-thirds (68%) of the total number of debugging barriers participants encountered during the course of our study.

4.4.1. Helping with Selection Barriers. *Selection* barriers were the most frequently encountered type of barrier; in fact, many participants became stuck in loops of repeated *Selection* barriers. Recall that this type of barrier reflects the difficulty of selecting the right words or messages in order to debug the intelligent assistant. This difficulty occurs because a text classifier often uses all words appearing in all emails as features, amounting to thousands of features. The sheer number of features makes finding a particular word in the representation of the assistant's logic time-consuming for a user. User-interface affordances can only do so much to alleviate this problem. For example, our bar graph was scrollable and allowed users to sort the features by weight or alphabetical order. Such interface tweaks are mere band-aids to the real problem—there are simply too many features for end users to consider.

One way to address the *Selection* barrier is to reduce the number of words displayed in the visualization by using a class of machine learning techniques known as *feature selection* [Guyon and Elisseeff 2003]. Instead of using all words in all emails as the set of features, we choose the top K most predictive words as the set of features. Feature selection in our email setting would reduce the number of features from thousands of words down to a few hundred. A particularly useful group of feature selection methods are based on sensitivity analysis, which is a technique used in statistical modeling to determine a model's robustness to changes in its parameters [Chan and Darwiche 2002].

Methods for sensitivity analysis are important for addressing *Selection* barriers, as they allow us to determine how much the weight of a word needs to be modified in order to change the probability that the email belongs to a folder F. Chan and Darwiche [2002] investigate the sensitivity of probabilistic queries on a Bayesian network in response to changes to a single parameter in the network. The authors then develop bounds on the effect of these changes to the query that allow the machine to determine if a change to a parameter for a word (i.e., the probability $P(W_i | F)$) has little to no effect on the predicted folder. These bounds can also be derived for the naive Bayes classifier because it is a special case of a Bayesian network. By omitting words that have little impact on the current prediction from the visualization would allow the user to focus attention on only those word weights that, if changed for this message, would alter the message's classification. Unfortunately, this approach only captures the effects of a single parameter change. Chan and Darwiche [2004] investigate the effects of changes to multiple parameters; but in the most general case this incurs a significant computational cost.

Although feature selection addresses some of the issues regarding selection barriers, it also introduces other problems. First, reducing the number of words from thousands to a few hundred still leaves the user with a fair number of features to sift through. Second, feature selection determines the top predictive features *using the training set*. If an email should be classified in a folder due to a distinct word that does not appear frequently in the training set, that word is unlikely to be among the top K predictive features. The end user would thus need to be able to add that word as a feature. Allowing end users to create new features is a promising direction for future work.

Furthermore, in the case of sensitivity analysis, even if we can determine the effects of changing multiple parameters, there are still an infinite number of parameter changes that can alter the current prediction to the desired folder. Sensitivity analysis is only useful for determining how much a change to the parameters will affect the final prediction. It is not helpful in identifying which changes out of a set of candidates would result in the most accurate classifier for future emails. The underlying issue to support the identification of features to modify is that there are an infinite number of combinations in which the weights on the words can be changed in order to cause the machine learning algorithm to classify an email to the desired folder. Ideally, these changes should improve the classification accuracy of the intelligent assistant not just for the current email message, but also future, as-yet-unseen messages. The distribution of future emails is clearly impossible—nevertheless, a possible method for helping with *Selection* barriers is to rely on the relationships between words in existing messages, in order to direct the end user's attention to particularly important words. These relationships would need to come from an external source, such as a commonsense database [Liu et al. 2003]. For example, if a message was misclassified to the *Finance* folder when it actually belongs in the *Sports* folder, a commonsense database could be used to identify words in the message that relate to sports and suggest that the user focus on these words. If, however, the end user's folder assignments for emails are obscure, ill, defined, or inconsistent, a commonsense database may be unable to help. Hence, identifying good features to modify remains a difficult open problem.

Simplifying the visualization by reducing the number of features needs to be carefully balanced against the stated design principles of our Why-oriented approach. In particular, we believe that the visualization should not obscure the logic of the intelligent assistant and that the user needs to be able to modify this logic to debug the assistant. Both of these principles are at risk if the visualization is overly simplified.

4.4.2. Helping with Coordination Barriers. *Coordination* barriers were the second most frequently occurring type of debugging barrier during our user study. Recall that these barriers concern the problem of how changing the assistant's logic to fix one prediction would influence the system's other predictions (i.e., coordinating how the system responds, or fails to respond, to the user's debugging efforts).

The "popularity effect" was the primary source of *Coordination* barriers: the folder with the largest number of filed emails dominated the classifier's predictions for the remaining messages in the inbox. This had the effect that a few participant changes could (and sometimes did) incorrectly reclassify dozens of messages into the newly popular destination folder. Unfortunately, participants whose strategy was to concentrate on fixing one folder at a time (making the folder of interest temporarily very popular) experienced the popularity effect again and again.

From a machine learning perspective, this popularity effect is primarily caused by the high dimensional nature of the data, the relatively sparse training data, and the class imbalance of the email folders. These factors cause the classifier to overfit both the training data and the participant feedback for the smaller folders. For example, suppose an end user employs the "one folder at a time" filing strategy. He is focused on putting messages into the *Systems* folder, which has keywords such as "Windows" and "McAfee" that are easily identified. Once a large number of messages have been filed to this dominant folder and the classifier learns from the newly acquired training examples, the distribution for the dominant folder will be accurately learned. However, the classifier is poorly trained on the nondominant folders. In fact, the classifier overfits the training data for the nondominant folders. Even worse, if the user jumps in to try to debug a nondominant folder by tweaking an email that belongs in it, the overfitting may be exacerbated more: the overfitting makes all other emails seem unlikely to be classified into the nondominant folders because they must match exactly the undersmoothed distributions for these folders. The classifier then incorrectly files almost all of the emails in the inbox under the dominant folder—the popularity effect thereby causing the user's valid correction to make the assistant worse.

On the surface it might appear that the solution from a machine learning perspective is simply to provide "sufficient" training data for all folders—but applying this solution to email classification is problematic. Many real-world email folders do contain small numbers of emails, resulting in sparse training data for email classifiers, and email is known to be "bursty," that is, emails from a small handful of folders dominate the inbox at certain times. Due to the imbalance in the number of emails belonging to each folder, explaining and addressing the popularity effect in domains such as these remains an open problem.

A second major source of *Coordination* barriers was the unexpected adjustment of decision boundaries resulting from participant feedback. ManiMatrix [Kapoor et al. 2010] tackles this problem by allowing end users to specify which portions of the boundary matter most to the user's task; it then adjusts the learning system's parameters to maximize the accuracy of the classifications the user cares most about. This ability could be beneficial for our Why-oriented approach as well; our prototype showed users the effects of their changes, but did not give participants the choice of specifying the effects they wished to see, and then adjusting the learning system's parameters to match the participant's goals.

4.4.3. Supporting Answers to the Why Questions Beyond Naive Bayes. In our study, we illustrated how one particular machine learning algorithm (naive Bayes) could be used to answer Why questions. Apart from naive Bayes, many other machine learning algorithms have been used to classify email messages. A natural question to ask is whether other machine learning algorithms can also provide easily understood answers to these "Why" questions.

Answering the "Why won't this message be filed in <Folder>?" question for folders f and f' requires computing $P(F = f \mid W_1, \ldots, W_m)$ and $P(F = f' \mid W_1, \ldots, W_m)$. Many machine learning algorithms can provide these probabilities. In fact, a large subclass of machine learning algorithms, known as *discriminative classifiers* [Ng and Jordan 2002], explicitly model $P(F \mid W_1, \ldots, W_m)$, and are well-suited to answer this "Why" question.

Some machine learning algorithms may be better than others at providing answers to certain Why questions. As an example, Bayesian networks provide a sophisticated mechanism for giving detailed explanations of how different pieces of evidence influence the final prediction made by the algorithm [Lacave and Diez 2002]; but this is computationally expensive. A current challenge for machine learning is to develop answers to Why questions of statistical machine learning algorithms that can be computed efficiently.

5. DISCUSSION AND CONCLUSION

This article presented a Why-oriented approach to support end-user debugging of intelligent assistants (in the context of a naive Bayes learning system for text classification), and explored the barriers and information needs participants encountered while debugging with a prototype instantiating our approach. Our contributions fall into three categories.

First, our Why-oriented approach is the first to bring the successful Whyline debugging approach to machine learning systems. Our approach is based on four underlying principles: (Representation-1) representations of both the assistant's logic (source code) and execution state must be available and manipulable by end users; (Representation-2) these representations must be faithful and transparent to the underlying logic (rather than attempting to hide some aspects for simplification); (ML-1) the explanations must be understandable to end users; and (ML-2) when the end user makes a correction, the learning system is required to honor it.

Using these principles, we developed an approach whose essence lies in allowing users to pose Why-oriented questions, answering them, and allowing users to change the answers, as follows.

—End users can ask why-questions of intelligent assistants (powered by statistical machine learning systems) to clarify the assistant's current behavior.
—The answers to these why-questions provide faithful explanations of the assistant's current logic and execution state.
—Users can debug their assistants interactively by modifying the answers to make them more correct (from the user's perspective), which immediately feeds adjustments back to the assistant's logic.

Discussion. The aspect of allowing users to change the answers (feature weights) is at heart closely related to "feature labeling," in which users specify logic by linking important features to a particular category, rather than the more commonly used "train more" route that amounts to providing multiple instances of additional training data. Our work is the first to allow users to freely specify which features matter and by how much—a direction we started in Stumpf [2007].

Measuring the effectiveness of such feature manipulation in our email study was not possible in a strictly objective fashion—given the participants' lack of history with the background conversations and cultural context, they had to make highly subjective judgments about how a message related to a topic. Thus, the "gold standard" (the folders assigned by the original authors of the messages') was not expected to consistently match our participants' opinions. (Indeed, our study's participants reported this issue explicitly; for example, "[I had to look] at previously filed messages in order to see what types of emails go where"). It was, however, possible to measure their subjective opinions on effectiveness. The participants' median response, when asked how successful they were at their tasks, was three on a five-point Likert scale, and was possibly affected by uncertainty as to whether their opinions of where emails should be filed were the "right" choices. Thus, to further investigate the issue of success, we recently ran a different study on a feature labeling mechanism in a less subjective domain [Wong et al. 2011]. The results were that end users who debuged by labeling features were significantly more accurate than users who labeled instances (i.e., the traditional "train more" approach).

Our second category of contributions lies in lessons learned for the human aspects of end users who debug intelligent assistants. Our exploratory study revealed that, although participants were able to use and understand our prototype, every participant encountered barriers while debugging their assistants. The two most common barrier types were *Selection* barriers, in which participants had difficulty in selecting the right features (words) or contexts (messages) to modify in order to correct the assistant; and *Coordination* barriers, in which participants wondered how the feedback they were about to give would change the assistant's other predictions or had trouble coordinating how the assistant responded (or failed to respond) to their modifications.

Discussion. The *Selection* barrier was particularly insidious, often leading to cyclical barrier patterns, and disproportionately affecting female participants. Theories such as the Selectivity Hypothesis [Meyers-Levy 1989] imply some broad-brush attributes for solutions that may help users overcome these barriers, suggesting that debugging tools should support both comprehensive and non-comprehensive debugging strategies, and that interfaces should adequately support a wide range of self-efficacy levels and attitudes toward risk.

Other, more specific, solutions to user barriers are suggested by the links our data showed between participants' barriers and the types of information they requested to overcome these barriers. For example, they verbalized the need for strategy ideas when faced with a *Design* barrier, and verbalized the need for help in selecting feature modifications that would cause particular classifications to change when faced with *Selection* barriers. Participants also expressed a preference for concrete, contextual explanations to overcome barriers; they behaved in a manner consistent with the idea of "active users" [Carroll and Rosson 1987], wanting solutions to the current problem (e.g., a misclassified message) rather than expressing an interest in the system's complete functionality.

The third category of contributions is a pair of new open questions for the machine learning aspects of supporting end-users in debugging their assistants: *how* exactly a machine learning system can provide the information described above to overcome the *Selection* barrier, and similarly, how can it provide the information to overcome the *Coordination* barrier.

Discussion. Regarding the *Selection* barrier, our participants made clear the information they wished they had—which words to concentrate on, how much to adjust each word, and what will happen next. Feature selection, such as through sensitivity analysis algorithms, may be a promising direction, but it raises as many issues as it solves. These issues may be partly addressable by allowing users to create new features and by incorporating knowledge of semantic relationships among words, for example, via a commonsense database. But despite these promising directions, identifying good features automatically for the user to modify remains a difficult open problem. Regarding helping the users to overcome *Coordination* barriers, many of which were tied to the "popularity effect," which arises in situations of class imbalance, as in email classification. Explaining and addressing the popularity effect in domains such as this one remains an open problem. Methods for allowing end users to specify the desired class distributions may be an alternative solution to this barrier.

In conclusion, we believe that end-user debugging of intelligent assistants is not only important, it is necessary—the end user is the only one with knowledge as to how his or her own specific adaptation of an assistant should behave. Our Why-oriented approach allows end users to debug such assistants by manipulating their underlying logic, not just their predictions. Our empirical results, based upon a prototype of the approach, identify barriers and information needs that end users encountered when debugging intelligent assistants, and also suggest solutions and open research questions as how to address these barriers and information needs. Ultimately, assisting end users to effectively debug their intelligent assistants opens new opportunities to achieve better and quicker adaptation on the part of the assistant, and thus better responsiveness to end-user preferences.

ACKNOWLEDGMENTS

We thank the participants in our study and Joe Markgraf, Amber Shinsel, Akshay Subramanian, and Rachel White for their assistance.

REFERENCES

Bandura, A. 1977. Self-efficacy: Toward a unifying theory of behavioral change. *Psych. Rev. 8*, 2, 191–215.

Becker, B., Kohavi, R., and Sommerfield, D. 2001. Visualizing the simple Bayesian classifier. In *Information Visualization in Data Mining and Knowledge Discovery*, U. Fayyad et al. Eds., 237–249.

Beckwith, L., Burnett, M., Wiedenbeck, S., Cook, C., Sorte, S., and Hastings, M. 2005. Effectiveness of end-user debugging software features: Are there gender issues? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 869–878.

Beckwith, L. 2007. Gender HCI issues in end-user programming. Ph.D. dissertation, Oregon State University, Corvallis, OR.

BURNETT, M., COOK, C., PENDSE, O., ROTHERMEL, G., SUMMET, J., AND WALLACE, C. 2003. End-user software engineering with assertions in the spreadsheet paradigm. In *Proceedings of the 25 International Conference of Software Engineering*. 93–103.

CARROLL, J. AND ROSSON, M. 1987. Paradox of the active user. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, J. Carroll Ed., MIT Press, Cambridge, MA, 80–111.

CHAN, H. AND DARWICHE, A. 2004. Sensitivity analysis in Bayesian networks: from single to multiple parameters. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*. 67–75.

CHAN, H. AND DARWICHE, A. 2002. When do numbers really matter? *J. Artif. Intell. Res.17,* 265–287.

CHEN, J. AND WELD, D. S. 2008. Recovering from errors during programming by demonstration. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*. 159–168.

COHEN, W. 1996. Learning rules that classify e-mail. In *Proceedings of the AAAI Spring Symposium on Machine Learning.*

COMPEAU, D. AND HIGGINS, C. 1995. Application of social cognitive theory to training for computer skills. *Inf. Syst. Res. 6*, 2, 118–143.

DAVIES, S. P. 1996. Display-based problem solving strategies in computer programming. In *Proceedings of the 6th Workshop for Empirical Studies of Programmers*. 59–76.

GLASS, A., MCGUINNESS, D., AND WOLVERTON, M. 2008. Toward establishing trust in adaptive agents. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*. 227–236.

GRIGOREANU, V., CAO, J., KULESZA, T., BOGART, C., RECTOR, K., BURNETT, M., AND WIEDENBECK, S. 2008. Can feature design reduce the gender gap in end-user software development environments? In *Proceedings of Visual Learning and Human-Centric Computing*. 149–156.

GUYON, I. AND ELISSEEFF, A. 2003. An introduction to variable and feature selection. *J. Mach. Learn. Res. 3*, 1157–1182.

HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. H. 2003. *The Elements of Statistical Learning*. Springer, Berlin.

KAPOOR, A., LEE, B., TAN, D., AND HORVITZ, H. 2010. Interactive optimization for steering machine classification. In *Proceedings of the Annual Conference on Human Factors in Computing Systems*. 1343–1352.

KISSINGER, C., BURNETT, M., STUMPF, S., SUBRAHMANIYAN, N., BECKWITH, L., YANG, S., AND ROSSON, M. B. 2006. Supporting end-user debugging: What do users want to know?. In *Proceedings of the Working Conference on Advanced Visual Interfaces*. 135–142.

KO, A. J. 2008. Asking and answering questions about the causes of software behaviors. Ph.D. dissertation; Tech. rep.CMU-CS-08-122, Human-Computer Interaction Institute, Carnegie Mellon University.

KO, A. J., MYERS, B., AND AUNG, H. 2004. Six learning barriers in end-user programming systems. In *Proceedings of Visual learning and Human-Centric Computing*. 199–206.

KONONENKO, I. 1993. Inductive and Bayesian learning in medical diagnosis. *Appl. Artif. Intell. 7*, 317–337.

KULESZA, T., WONG, W., STUMPF, S., PERONA, S., WHITE, R., BURNETT, M. M., OBERST, I., AND KO, A. J. 2009. Fixing the program my computer learned: Barriers for end users, challenges for the machine. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*. 187–196.

LACAVE, C. AND DIEZ, F. 2002. A review of explanation methods for Bayesian networks. *Knowl. Eng. Rev. 17*, 2, 107–127.

LIEBERMAN, H. (ED.) 2001. *Your Wish is My Command: Programming By Example*. Morgan Kaufmann.

LIM, B. Y., DEY, A. K., AND AVRAHAMI, D. 2009. Why and why not explanations improve the intelligibility of context-aware intelligent systems. In *Proceedings of the Annual Conference on Human Factors in Computing Systems*. 2119–2128.

LIM, B. Y. AND DEY, A. K. 2009. Assessing demand for intelligibility in context-aware applications. In *Proceedings of the 11th International Conference on Ubiquitous Computing*. 195–204.

LITTLE, G., LAU, T., CYPHER, A., LIN, J., HABER, E., AND KANDOGAN, E. 2007. Koala: Capture, share, automate, personalize business processes on the web. In *Proceedings of the Annual Conference on Human Factors in Computing Systems*. 943–946.

LIU, H., LIEBERMAN, H., AND SELKER, T. 2003. A model of textual affect sensing using real-world knowledge. In *Proceedings of the International Conference on Intelligent User Interfaces*. 125–132.

MARON, M. E. 1961. Automatic indexing: An experimental inquiry. *J. ACM* 8, 3, 404–417.

MCDANIEL, R. AND MYERS, B. 1999. Getting more out of programming-by-demonstration. In *Proceedings of the Conference on Human Factors in Computing Systems*. 442–449.

MEYERS-LEVY, J. 1989. Gender differences in information processing: A selectivity interpretation. *Cognitive and Affective Responses to Advertising,* P. Cafferata and A. Tybout Eds., Lexington Books.

MYERS, B., WEITZMAN, D., KO, A. J., AND CHAU, D. H. 2006. Answering why and why not questions in user interfaces. In *Proceedings of the Conference on Human Factors in Computing Systems*. 397–406.

Ng, A. Y. and Jordan, M. I. 2002. On discriminative vs generative classifiers: A comparison of logistic regression and naïve Bayes. *Advances Neural Inf. Process. Syst. 14*, 841–848.

Patel, K., Fogarty, J., Landay, J., and Harrison, B. 2008. Investigating statistical machine learning as a tool for software development. In *Proceedings of the Annual SIGCHI Conference on Human Factors in Computing Systems*. 667–676.

Poulin, B., Eisner, R., Szafron, D., Lu, P., Greiner, R., Wishart, D. S., Fyshe, A., Pearcy, B., Macdonell, C., and Anvik, J. 2006. Visual explanation of evidence in additive classifiers. In *Proceedings of the AAAI Spring Symposium*. 1822–1829.

Russell, S. J. and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, NJ.

Stumpf, S., Rajaram, V., Li, L., Burnett, M., Dietterich, T., Sullivan, E., Drummond, R., and Herlocker, J. 2007. Toward harnessing user feedback for machine learning. In *Proceedings of the International Conference on Intelligent User Interfaces*. 82–91.

Stumpf, S., Sullivan, E., Fitzhenry, E., Oberst, I., Wong, W.-K., and Burnett, M. 2008. Integrating rich user feedback into intelligent user interfaces. In *Proceedings of the International Conference on Intelligent User Interfaces*. 50–59.

Stumpf, S., Rajaram, V., Li, L., Wong, W.-K., Burnett, M., Dietterich, T., Sullivan, E., and Herlocker, J. 2009. Interacting meaningfully with machine learning systems: Three experiments. *Int. J. Human-Comput. Stud. 67*, 8, 639–662.

Subrahmaniyan, N., Kissinger, C., Rector, K., Inman, D., Kaplan, J., Beckwith, L., and Burnett, M. 2007. Explaining debugging strategies to end-user programmers. In *Proceedings of the Visual Learning and Human- Centric Computing*. 127–136.

Subrahmaniyan, N., Beckwith, L., Grigoreanu, V., Burnett, M., Wiedenbeck, S., Narayanan, V., Bucht, K., Drummond, R., and Fern, X. 2008. Testing vs. code inspection vs. ... what else? Male and female end users' debugging strategies. In *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems*. 617–626.

Talbot, J., Lee, B., Kapoor, A., and Tan, D. S. 2009. EnsembleMatrix: Interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*. 1283–1292.

Tullio, J., Dey, A., Chalecki, J., and Fogarty, J. 2007. How it works: A field study of nontechnical users interacting with an intelligent system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 31–40.

Ungar, D., Lieberman, H., and Fry, C. 1997. Debugging and the experience of immediacy. *Comm. ACM* 40, 4, 38–43.

Vander Zanden, B. and Myers, B. 1995. Demonstrational and constraint-based techniques for pictorially specifying application objects and behaviors. *ACM Trans. Comput.-Human Interac.* 2, 4, 308–356.

Wagner, E. and Lieberman, H. 2004. Supporting user hypotheses in problem diagnosis on the web and elsewhere. In *Proceedings of the International Conference on Intelligent User Interfaces*. 30–37.

Wong, W.-K., Oberst, I., Das, S., Moore, T., Stumpf, S., Mcintosh, K., and Burnett, M. 2011. End-user feature labeling: A locally-weighted regression approach.In *Proceedings of the International Conference on Intelligent User Interfaces*. 115–124.

Received March 2010; revised February 2011; accepted March 2011