The Do-I-Care Agent: Effective Social Discovery and Filtering on the Web

Mark S. Ackerman[1]
Brian Starr
Michael Pazzani
Information and Computer Science
University of California, Irvine
{ackerman, bstarr, pazzani}@ics.uci.edu

ABSTRACT

The Web is a vast, dynamic source of information and resources. Because of its size and diversity, it is increasingly likely that if the information one seeks is not already there, it will be soon. Unfortunately, finding the right places to look, and persistently revisiting those places until the information is available is an onerous task. In this paper, we describe Do-I-Care (DICA), an agent that uses both technical and social mechanisms to ease the burden of locating "interesting" new information and resources on the Web.

DICA monitors Web pages previously found by the agent's user to be relevant for any changes. It then compares these changes against a user model, classifies them as potentially interesting or not, and reports the interesting changes to the user. The user model is derived by accepting relevance feedback on changes previously found. Because the agent focuses on changes to known pages rather than discovering new pages, we increase the likelihood that the information found will be interesting.

DICA combines an effortless collaboration mechanism with the natural incentives for individual users to maintain and train their own agents. Simply by pointing DICA agents at other agents, changes and opinions can be propagated from agent to agent automatically. Thus, individuals train and use DICA for themselves, but by using a simple technical mechanism, other users can use those results without the additional effort that often accompanies collaboration.

KEYWORDS: computer-supported cooperative work, CSCW, social filtering, collaboration, World Wide Web, machine learning, agents

[1]Author to contact: Mark S. Ackerman, Information and Computer Science, Computer Science 444, Irvine, CA, 92697, USA. ackerman@ics.uci.edu. Telephone: 714-824-7355. Fax: 714-824-4056.

1. INTRODUCTION

After continual, exponential growth, the World Wide Web has become a vast information resource. Virtually any information can be found there eventually, if only one knows where to look for it. Unfortunately, the Web's vastness and rate of growth, while ensuring the variety and comprehensiveness of information that could *theoretically* be found (given an infinite amount of time, resources, and motivation), make it increasingly unlikely that any *particular* piece of information will be found with a reasonable expenditure of effort. Yet, one is often most interested in finding the "right" information, rather than just generally relevant information.

For example, you might want to know:

- When a colleague has an interesting new paper.
- About new, interesting jobs.
- When friends are on a Web-based chat system.

In each of these situations, the user has a known information need and will wish to monitor a known information source for new information. However, he is probably not interested in all potential changes to that source. Most users would prefer to see only the *interesting* changes. One does not wish to know about formatting changes to a colleague's "new papers" Web page. Similarly, one probably wants to know about interesting jobs in a specific area, such as graphics or knowledge engineering, rather than any job, such as secretarial or accounting positions.

It would also be valuable to re-use others' efforts. One may wish to look for a graphics position in the San Mateo area, and using another user's pre-existing agent would be beneficial. Similarly, a graduate student may wish to specialize a professor's "interesting papers" agent.

Our Do-I-Care Agent (DICA) attempts to solve both of these problems. DICA examines known Web pages for interesting changes as specified by a user-trained model. Specific DICA agent's results can be shared through a simple and straight-forward mechanism.

Before providing the details of DICA's user interface, machine learning, and collaborative architecture, we first survey existing mechanisms for finding information on the Web.

2. RESOURCE DISCOVERY SYSTEMS

Tools to discover or filter information on the Internet are known as resource discovery systems (Schwartz et al. 1992). The most common types of systems used for locating new information on the Web are active browsers, index creation and lookup systems, generic Web change notifiers, and social filters.

2.1. Active browsers

Active browser systems, such as Fish Search (DeBra and Post 1994) and WebWatcher (Armstrong et al. 1995), treat the Web as a graph that can be searched using evaluation functions and heuristics derived from a model of users' goals and interests. While users browse the Web, Web Watcher offers navigational suggestions that direct them along paths that maximize these goals. User models are obtained directly from users while they are browsing and require users' active participation. If the user model is accurate, the advice given guides users to useful Web locations.

By actively involving users in the search, active browsers obtain immediate feedback about their suggestions. This allows agents to quickly improve their user model, and users to familiarize themselves with and better exploit the information space and capabilities of their agents. However, because searching is in real time, given the size and complexity of the Web, agent suggestions can approximate hill climbing. Thus, the choice of starting points is particularly important, and some form of preliminary catalog search may be necessary. More seriously, active search requires human participation, time, and attention, which are generally scarce resources.

A variation described by Balabanovic and Shoham (1995) is an agent that wanders the Web independently from the user. It reports interesting pages to the user, who can then rate these suggestions. Therefore, the system uses relevance feedback, yet only occasional human participation is required. However, this agent is optimized for examining new pages, not for detecting changes to known sources of information.

2.2. Index creation and lookup

The second type of technical resource discovery systems, index creation and lookup systems such as ALIWEB (Koster 1994), Harvest (Bowman et al. 1994) and Lycos, utilize an information retrieval metaphor where the Web is treated as a database of information to be cataloged and indexed.

These systems make the Web accessible to users who do not need to know its organization. However, users must articulate their information goals in well-formed queries. Also, due to the Web's rapid, uncoordinated growth, precision is increasingly important (Pinkerton 1994); these systems have the standard information retrieval problems with precision.

Finally, users interested in new information in the future must periodically resubmit their queries if notification is not supported. Some systems, such as the SHADE and COINS matchmaker systems (Kuokka and Harada 1995), utilize knowledge sharing to match user requests with information provider's "advertisements", and can notify users when they receive updates from providers about interesting changes. However, these systems rely on provider honesty, since there are no technical checks against suppliers that "cheat" (Brown et al. 1995). For example, providers may claim they have more interesting information than they actually do, hoping that requesters settle for what is available rather than spend additional resources locating better information elsewhere.

2.3. Generic Web change notifiers

There are also several systems that monitor Web pages for *any* modification, such as URL-minder (1996) and WebArranger. However, the user must still discover the Web sites to be monitored. More importantly, many of the changes detected may be irrelevant, and each change must be examined by the user.

2.4. Social resource discovery

While technical discovery methods attempt to automate the discovery of new information, they neglect perhaps the single most important method of discovery upon which people rely -- other people. Social filtering systems attempt to exploit social networks to find relevant information.

Regardless of one's particular information need, there are usually others that are both knowledgeable about the subject domain and skillful enough at searching to find useful new information. These people, with their relevant expertise and interests, may be better at finding and selecting useful information than any technical mechanism. One would like to gather up all the relevant information found by others and simply select the choicest bits.

The goal of social filtering systems, then, is to aggregate and share the fruits of individual activity and knowledge. Social filters, also called social resource discovery systems, facilitate the dissemination of useful information from people who are good at finding things in a particular field to those who are not. Social filtering systems utilize the opinions of others to select only the most valuable information from existing collections.

Relatively few social resource discovery systems currently exist. The Pointers system (Maltz and Ehrlich 1995) facilitates the distribution of links to resources with accompanying context. This allows users to point out new and interesting information to one another. Nonetheless, while the benefits to a pointer's recipient seem clear, the system relies

on a provider's desire to be helpful that may not always exist. Furthermore, providing information to others using the system requires an active effort.

Both Ringo (Shardanand and Maes 1995) and GroupLens (Resnick et al. 1994)use correlations among user profiles to determine interest for a user in a new piece of information. GroupLens uses others' newsgroup article ratings to predict an article's relevance to a given user. Ringo, and its successor Firefly, provide film and music ratings based on the correlations between similar users. Both systems provide some benefits for users to offer opinions, since the weights assigned to others' opinions are proportionate to their past correlation with the user's opinions. However, in both systems, the emphasis is on providing collaborative effort for unclear personal gain. Furthermore, the quality of others' opinions may also be unclear; sometimes, one wants the opinions of experts in a field.

A significant problem with collaborative discovery and filtering of Internet resources, such as interesting Web pages, is that it requires work by other users. Unfortunately, users often will not do the extra, altruistic work -- especially when the users are busy experts. This can lead to adoption failures, since few users will pay upfront costs for uncertain benefits. DICA attempts to alleviate the cost/benefit imbalance by exploiting naturally occurring individual effort in a way that facilitates information sharing without incurring any significant additional costs. It is similar to Answer Garden (Ackerman 1993) in its use of individual motivations and incentives resulting in collective activity.

In summary, there are a plethora of ways to find valuable bits of information on the Web. There exist many resource discovery and filtering tools that help one locate new sources of information. No existing system, to our knowledge, re-discovers *interesting* information on *known, previously discovered* sources. Furthermore, no system does this by leveraging both technical and social discovery mechanisms. The next section provides an overview of DICA, our attempt to address these needs.

3. DICA: THE DO-I-CARE AGENT

Do-I-Care was designed to help users discover *interesting* changes on the Web, using both technical and social means. DICA agents automate periodic visits to selected pages to detect interesting changes on behalf of individuals. They use machine learning to identify by example what changes are interesting and how often they occur. It is in users' best interests to keep their personal agents informed about relevant pages and about the quality of reported changes.

Because different pages may be interesting for different reasons, users may maintain multiple Do-I-Care agents specialized on particular topics or kinds of Web pages. For example, the criteria for an interesting journal announcement may be different from that of a news item.

4. SYSTEM ARCHITECTURE

Each Do-I-Care agent must:

- Periodically visit a user-defined list of target pages, and identify any changes since the last time it visited.
- Rank incoming changes according to how interesting they are likely to be.
- Notify the user about interesting changes or events.
- Accept relevance feedback on the interestingness of the past changes.
- Allow user customization of all important features.
- Facilitate information sharing and collaboration between individuals and groups.

We cover each of these in turn.

4.1. Revisiting target pages

We assume that users have lists of Web pages that will have interesting information. For example, a user might have bookmarks pointing to travel-related Web pages. If the user would like to know when information about new holiday destinations are posted, he can direct a DICA agent to periodically monitor these pages for any changes. By limiting the search to a small subset of Web pages specifically selected *because* they are likely to contain interesting information, we can greatly improve precision. On the other hand, our agent will not attempt to find new pages for the user, although, as we will see, he may learn about new pages from other users.

Each agent maintains a list of URLs that it is monitoring. These URLs can point to any Web page reachable by the user's favorite browser, including gopher menus, file lists, and the output of CGI scripts. For each URL, the agent keeps track of the last time it was visited, how frequently it should be visited, how frequently it actually changes, and how often the agent has been unable to read the URL's page. The last two allow the agent to adjust its visitation frequency and notify the user if there is a problem with the URL, respectively.

Periodically, a task runs that scans the agent's list of URLs, and places all of those due for a visit on a queue. Separate tasks are then spawned for the first n queued URLs, where n is a configurable option, and the associated HTML source is read using standard HTTP protocols.

4.1.1. Detecting changes in web pages

DICA compares each Web page with the previous contents of that page. The periodicity is set by the user or learned by the system.

DICA assumes that Web pages will change incrementally, and that the user is only interested in additions or changes. These are reasonable assumptions. Once a Web page stabilizes, authors seldom radically change them. Change is usually incremental, and new things are usually added in ways that are consistent with what has been done in the past. This is true because of the additional work involved in radical alterations and because authors tend to follow styles consistent with the other pages of similar content and purpose. In a public environment like the Web, standard forms and genres become socially constructed over time (Berger and Luckmann 1966).

Because pages usually change incrementally, and changed segments are usually contiguous, the quality of the changes we can detect and the ease by which we detect them are greatly improved. Because changed segments are usually devoted to a particular topic, the efficacy of the user's relevance feedback is also increased. However, when a single detected change is likely to contain many different topics, then the meaning of any feedback the user might provide becomes unclear at best.

To compare page versions, DICA uses the Unix utility "diff" together with additional pre- and post- processing. All runs of whitespace in downloaded HTML source are replaced by a single space, then a carriage return is inserted before and after significant HTML tags (such as anchors or paragraphs). When the resulting text is compared by diff, diff reports changes in segments delineated by tags rather than lines of text. This method works well with actual HTML documents because it tends to group all the text and tags related to a particular topic together in one change. The page reading task then returns a list of those segments that were added or changed since the last time the page was read, plus the segment number of the beginning of each change. The latter allows DICA to later identify where the change occurred on the original page. DICA currently ignores deletions reported by diff.

4.2. Deciding whether a change is interesting

The output of the change detection module is essentially a stream of mini-documents, where each mini-document contains the full text of the change detected. These documents are then rated by a first-order Bayesian classifier (Duda and Hart

1973), and the user is notified about changes where the scores exceed a configurable threshold. This classification step is the heart of DICA.

The Bayesian classification process consists of two steps, profile building and scoring. Both profile building and scoring also include an attribute acquisition step. We begin here with an intuitive explanation of how first-order Bayesian classification works, in order to motivate a more complete discussion of profile building and scoring later.

4.2.1. Bayesian classification

Bayesian classification assumes that for every classifiable object, there exists a set of computable attributes that together allow the calculation of probabilities that the object belongs to each of a known set of categories. For example, if one is trying to classify a vehicle as either a car or a motorcycle, and one knows the vehicle has 4 wheels, one can say that the probability that the vehicle is a car is high, and the probability that it is a motorcycle is low, because most of the observed cars have 4 wheels, while most of the motorcycles have 2.

This is not to say that a car *has* to have 4 wheels, nor that every 4 wheeled vehicle we see *will* be a car. Simply by inspecting the value of this one attribute, we gain some predictive power. We can improve the accuracy of the prediction by looking at multiple attributes. For example, we can ask if our vehicle has a round steering wheel, or 4 hubcaps. Each attribute contributes some classificatory probability. For example, if the vehicle has 4 wheels, the probability is 95% that it is a car, and if it has a round steering wheel, the probability is 99%; yet, if both are true the probability increases to 99.93% (assuming 2 possible classifications and independence).

Actually, the situation is more complicated, since not all attributes change independently of each other. Most vehicles that have 4 wheels *also* have 4 hubcaps, so once we know that a vehicle has 4 wheels, the fact that it also has 4 hubcaps does not increase our certainty that it is a car. Although non-independent probabilities are a source of error classifications, the amount is usually relatively small and higher-order Bayesian classifiers are prohibitively expensive.

Furthermore, many features that one could potentially examine have little informative value for classification purposes. For example, the fact that a vehicle has an engine does not help one decide whether it is a car or a motorcycle, since both types of vehicles are equally likely to have one. One the other hand, if a vehicle has handlebars, it is almost certainly a motorcycle and almost certainly not a car. Because calculating attribute probabilities is not free, and using the wrong attributes might introduce "noise" and make categories harder to learn, one would prefer to use only the most informative attributes for classification purposes.

Finally, when dealing with finite sample sizes, random variations in the examples can introduce noise into classifications. For example, say that the classifier happens to see only red motorcycles and green cars. From this, it may erroneously conclude that color is an excellent way to discriminate between the two types of vehicles. Unfortunately, we know from our experience that differently colored cars and motorcycles do exist, and that color has almost no informative value. Given a large enough sample size, our classifier will eventually learn this, but perhaps only after many incorrect predications.

In summary, first-order Bayesian classifiers estimate the probability that an object will belong to a class, where these estimates are determined from the past probabilities of objects that share attributes with the same values as those of the given object. In practice, it is better to choose attributes that are independent, informative, and not subject to random variation. Fortunately, the impact of imperfect features tends to decrease as the user trains the classifier (discussed below).

4.2.2. Determining attributes

Since the ratings given by DICA's classifier depend solely on the attributes it extracts for each document, the choice of attributes is important. As with many information retrieval and filtering systems, the discriminatory attributes considered by DICA are derived from keywords extracted from the document's text. The entire document is parsed into a list of unique words, where a word is a sequence of alphas (a-z and A-Z) delimited by a non-alpha on either side. HTML tags (besides anchors) and duplicate occurrences of words are ignored.

Unlike some systems, DICA performs almost no translation on these extracted keywords. In particular, we chose not to tag these keywords with parts of speech, stem them, or translate them into common concepts using a thesaurus. These are clearly next steps for a production-quality agent.

4.2.3. Creating a classification profile

A classification profile is a list of feature values (in DICA's case, the presence or absence of words in a document), and the corresponding probabilities that documents having those values belong to one or more classes. To create a profile, a set of user classified documents that have been reduced to their corresponding list of attributes is needed. In particular, DICA uses three sets of word lists: those representing "interesting", "uninteresting", and "never-show" documents.

To do this, DICA first constructs a master list of all unique words present in at least one document, sorted according to their informativeness. DICA determines how informative a word is by computing the mutual information (Quinlan 1986) between the presence of a word in previously classified interesting, uninteresting and never-show changes. From these, DICA

then arbitrarily chooses the top half, and discards the rest. This usually rejects most "noise" words (like "and" and "the"), and users can explicitly reject any that remain by adding them to a stop list. For each remaining word, DICA then estimates two probabilities for each of the three classes: out of all the documents classified as interesting (or uninteresting or never-show), what percentage of them contain (or do not contain) the word in question? DICA also calculates the probability that any random document will be interesting, uninteresting or never-show.

Finally, because users might wonder why DICA rates changes as it does, DICA identifies the top n keywords that, if found in a document, most increase the probability that the document will be interesting. DICA sorts these words according to the probability that they will be present in an interesting document multiplied by the probabilities that they will be absent from an uninteresting or never-show document. These words are listed at the top of the agent's web page, and they are highlighted in displayed changes. Since user tastes and agent requirements differ, the number n is user-configurable as will be explained in the section on configuration options.

4.2.4. Classifying documents

Newly discovered changes are parsed to extract their attributes. Then, DICA estimates the probability that they are interesting, uninteresting, or never-show based on the previously calculated profile as follows:

$$Po(class \mid doc) = P(class) * \prod_{wp \in w} P(class \mid w) * \prod_{wa \in w} P(class \mid \sim w)$$

where:

doc = the document we are currently rating.

P(class | doc) = Probability that this document belongs to class

P(class) = Prior probability that any given document might belong to class

P(class | w) = Probability that a word will be present in a document if the document belongs to class.

P(class | ~w) = Probability that a word will not be present in a document if the document belongs to class.

wp = set of all informative words present in this document

wa = set of all informative words absent from this document

Then, since class* encompasses all possible hypotheses (i.e., possible categorizations for the document), we then normalize P(class | doc) so that the probabilities of all classes in class* sum to 1:

$$P(class \mid doc) = P_O(class \mid doc) / \Sigma_{c \in class^*} P(doc \mid c)$$

class* = set of all possible classes (interesting, uninteresting, never-show)

`P(interesting | doc)` is then the probability that a document will be interesting, ranging from 0 to 1. This value is compared to a user-configurable threshold to determine whether a document is sufficiently likely to be considered interesting by the user. Those documents scoring greater than this threshold are reported to the user and to any listening agents.

`P(never-show | doc)` is the probability that a document should never be shown to users. If this value is greater than a user-configurable threshold, then DICA will not inform users of its existence, since the document is so unlikely to be interesting that it would be a waste of time for users to view it. Since users do not rate changes as never-show (since they are not shown to the user), the never-show category should shrink over time relative to the other two.

4.3. Notifying the user about changes

There are two ways that DICA informs users about interesting changes or events: through a user-configurable list of e-mail addresses and through the agent's Web pages. As we will see later, an agent's Web page also serves as the agent's user interface, as well as the conduit by which the agent communicates with other agents.

Because users have only a finite amount of time and attention, DICA's notification strategy must avoid being intrusive and burdensome, yet remain timely and informative. If DICA sends notifications too frequently, users will eventually start to ignore, or even resent, DICA. However, if DICA fails to notify users when warranted, they may lose faith in DICA's judgment. In both cases, faulty notification is almost worse than none, since it can either add extra work or cause users to miss important information.

For this reason, the conditions under which DICA will send e-mail is user-configurable. The user can chose to receive e-mail when any change besides a never-show is detected, only when interesting changes are detected, or not for any kind of change. In the first two cases, all the changes that occur in a particular agent run are grouped into one e-mail message. The sender of the e-mail is the agent, while the e-mail's return address is user-configurable (usually set to the owner of the agent, in case more than one user is being sent notifications). The subject of the e-mail indicates whether or not any interesting changes were identified. The body of the message starts with the URL of the agent's main Web page, described below. If no interesting changes were identified, the body of the message simply states that some changes were found, but does not list the changes.

However, if interesting changes were found, the agent lists the URLs of each change, followed by a list of those keywords in the change that caused the agent to find it interesting. This is often sufficient to give users a rough idea what the

change is about without overwhelming them with information. If users wish to see the complete text of a change, they can then follow the URL at the top of the message to the agent's Web page.

Each agent actually has two Web pages: the main page, and the configuration page. The main page, which is reached via the agent's URL, contains a subset of the changes detected by the agent, selected according to user-definable criteria. The main page also contains a link to the agent's configuration page, which will be discussed in greater detail below.

An agent's main page is where users can both view and rate the changes detected by that agent. To help the user understand what types of changes this agent classifies as interesting, the agent lists the most highly ranked keywords at the top of the page. (The number is user-configurable.) Following are those changes that the user has not yet rated (nor presumably seen), then a user-definable subset of those changes he has already seen.

Because users need to know varying information about a change, and screen space is valuable, DICA attempts to display the minimum needed for each change, while allowing users to adjust this as desired. All displayed changes include a detection date and a link to the page on which the change was found. The link will display the changed page where the change occurred, highlighting the change. The change text is pretty-printed, preserving all the links and images of the original. Additionally, the most highly interesting keywords in the change are highlighted. Finally, because users might browse multiple changes in one session, following any of the above links cause a new browser window to be opened, so the agent's main page also remains visible.

The agent normally controls which changes will be displayed and in what order, so as to make interesting changes more visible and accessible. Unrated changes (i.e., those not rated by the user) are displayed before rated changes. Rated changes are then sorted by whether or not the user rated them interesting. All changes can then be sorted by how "interesting" the agent found them; the user can optionally disable this for unrated changes so that related changes remain adjacent. All changes are finally sorted by the date and time they were detected.

By default, all unrated changes are "shown", meaning that the entire contents of the change are shown. Changes that have been rated are "not shown", meaning that only the date, URL, rating, and keywords are shown. The user can individually fold and unfold changes simply by changing the corresponding menu selection for that change and pressing the "register selections" button. Finally, if the user chooses, the agent will avoid mentioning rated changes that were rated uninteresting by both the agent and the user.

4.4. Accepting relevance feedback

As described above, incoming changes are automatically classified by the agent. However, the user is ultimately the final judge of whether the change was truly interesting or not. By assigning ratings to detected changes, the user provides training examples that improve the accuracy of the agent's future predictions, as well as the usefulness of its change notifications. Since users provide relevance feedback, we can view the process of determining whether a change is interesting as a classification task and the process of learning a user profile as a supervised learning task.

In order to make the user's training task as painless as possible, DICA has been designed to simplify and facilitate reading and rating changes. As previously noted, each agent has an associated main Web page to report detected changes to the user. This page also serves as the means by which users can provide relevance feedback. Changes on this page are sorted according to how interesting they are likely to be, with more interesting changes near the top. Because of this, users are often able to minimize the number of changes they actually examine, because at some point in the page no further interesting changes are likely to be found. The mere fact that DICA has predicated a change's rating is a helpful guide to how interesting the change is likely to be.

Once a user decides whether a change is interesting, DICA also tries to simplify the training process for the user. Every listed change has an associated menu with four choices: "I Care", "I Don't Care", "Never Show", and "Unrated". All changes are initially "Unrated". Originally, the only way users could assign ratings was by selecting the appropriate choice and pressing the "register choices" button at the top of the page. This required a separate action for each change and was too much work. Fortunately, we noticed that after a short training period, DICA would do a fine job predicating what a user's choice would be, and so we created a shortcut button. If the user presses the "as suggested" button, any remaining unrated changes default to the rating recommended by DICA. Now, the user needs only quickly scan the list of changes for bad predictions, and leave the rest unrated. This feature helps lower the cost for the user to train his agent.

Since untrained agents cannot accurately predict user ratings, two additional short-cut buttons are provided: one to assume unrated changes are uninteresting, and another to assume unrated changes are interesting. By using either of these two buttons, the user will only need to hand rate half of the changes in the worst case.

Another way user reading and rating is facilitated is through the never-show classification. We have found that some types of changes are obviously uninteresting, yet common enough to be a nuisance at best and overwhelming at worst. For example, many Web pages tend to periodically update the page's modification date even when nothing of consequence was

actually changed. Although DICA is usually quickly able to distinguish such changes from truly interesting ones, there is simply no reason why the user should ever have to see them at all.

If the user identifies an unrated change as never-show, not only is this change never displayed again, but neither are similar changes in the future. In fact, DICA will never suggest that a change be rated never-show; it will not display the change to be rated. Therefore, unlike interesting and uninteresting changes, few changes are ever actually rated never-show which means that the range of changes covered by the never-show category tends to shrink over time relative to the other two. Periodically, "worthless" changes will start to slip past the never-show category again, and it must be retrained. This property is intentional, because it would be equally bad to withhold too much from the user.

Once the users have recorded their ratings, they can be used to improve agents' classification accuracy. The next time the agent runs, the user's selections will be incorporated into a revised profile. (However, since users interests may change over time, DICA can also be programmed to automatically "forget" about old ratings after a user-definable period of time.) The revised filter is then used to rate new changes -- existing agent suggestions are not affected.

4.5. Agent configuration and customization

Throughout the description of DICA's functionality, we have pointed out many of the ways that the agent's behavior can be customized to match both the user's tastes and the task at hand. In this section, we will summarize the most important customization and configuration options that are available, and provide some typical usage scenarios.

4.5.1. Choosing the sites to monitor

Perhaps the single most important customization option is the ability to provide a list of Web pages to monitor. Not only does user's choice of Web pages determine the subject matter of the changes the agent will detect, but it also determines their quality and frequency. The choice of Web pages also strongly influences the efficacy of the agent's eventual filtering and training.

Different kinds of Web pages have different "personalities." DICA does best with pages where small portions are frequently added or deleted. An example of such a page is one that consists of USENET newsgroup subject lines (available through the Web). DICA also seems to perform well on many on-line newspapers. Even though they change completely almost every day, they retain their overall structure; therefore, DICA is able to pick out the individual news stories.

DICA has more trouble with some Web pages that tend to change very seldom, but when they do change, huge portions of the page change at the same time. Since the changes detected on this type of page tend to be large and mix many different topics, classified examples appear "noisy," degrading training. We are currently investigating how to correct this.

4.5.2. Options dealing with detected changes

An option which influences both the training efficacy and the readability of detected changes is the choice whether to consolidate all changes detected on a Web page into one change per visit. For example, if three distinct changes were detected in a page, they could be merged and reported to the user as one large change. This option is useful if the pages tend to have many small distinct changes that are related. Merging the changes, then, makes the result more comprehensible to user, as well as gives the classifier more with which to work. Conversely, setting this option would not be desirable if the changes tend to be about different topics, since this could reduce classification effectiveness.

A second change option controls the sorting order of unrated changes. Normally, all changes, rated or not, are sorted according to how likely they are to be interesting. Unrated changes are moved to the top, then changes the agent found more interesting are moved ahead of less interesting ones. Finally, changes are sorted according to their order of discovery. This works well if changes can be understood independently of each other, but is less desirable if topics tend to be spread over move than one change. By selecting "only sort unrated by date", unread changes found nearby in the same document are listed together on the agent's main page.

Finally, in order to make DICA's classification scheme more transparent and help users locate points of interest in detected changes, DICA can optionally use "hot words" to annotate changes and notifications. As previously mentioned, hot words are those words that, if present in a change, most imply that the change was interesting. Users' ability to quickly skim changes can be improved if the embedded hot words are highlighted. Also, the hot words in a change can provide a useful summary in cases where the full text cannot be shown (e.g., e-mail notifications). Finally, as their agents' hot words approach the users' own list of "keywords", users can judge the efficacy of their agents' training process. However, the number of hot words used is a tradeoff: More hot words can be distracting, and might cause uninformative or misleading hot words to be used if the agent is imperfectly trained. Because of this, the number of hot words used is configurable.

4.5.3. E-mail options

Because DICA needs to keep users informed without overwhelming them, its e-mail notification strategy is user-configurable. First of all, users identify to whom an agent will send mail by explicitly adding the e-mail addresses to a list.

The list can be as long or short as desired -- if it is empty, the agent will not e-mail notifications. The list can contain any valid e-mail address, potentially including newsgroups and mailing lists.

Secondly, users can chose the circumstances under which DICA will send notifications. Most users prefer to be notified only when an interesting change was found, so they know when to visit their agent's Web page. However, because untrained agents consider all changes uninteresting until they have seen some examples of interesting changes, users often find it useful to initially request notification about both interesting and uninteresting changes until their agents are working reliably. Of course, some users prefer to periodically visit their agents' Web page on their own, and would prefer not to receive any notification. DICA can accommodate this also.

Additionally, DICA has options for handling bad URLs. Most users familiar with the Web realize that URLs can "go bad" from time to time, and would appreciate notification when this occurs so they can take remedial action. However, because transitory failures occur often, even if a Web page was unreadable once, it might still be reachable later. Therefore, by default, DICA will only send a bad URL notification after five successive failed attempts at different times. This ensures that most notifications are legitimate, but if URLs are expected to be bad often, DICA can optionally never send any bad URL notifications. Finally, if URLs are always expected to be good, URLs are changing often, or one is otherwise unsure about them, DICA can optionally provide immediate notification whenever any bad URL is detected.

Finally, users can chose the return address of the agent's e-mail. This is useful if the agent is configured to send e-mail to multiple recipients, so users can send comments and problem reports simply by replying to an agent's message.

4.5.4. User defined stop words

Because our agent was designed to work on any document accessible via the World Wide Web, we made as few assumptions as possible about the contents of a change. Currently, we assume changes are composed of alpha (i.e., a-z and A-Z) sequences separated by non-alphas and tags. We also assume that non-alphas and tags are not significant discriminators of meaning. Our decision to ignore the contents of tags and non-alpha symbols restricts and biases the categories we can recognize. (For example, we cannot identify boldfaced text or advertisements about homes under $100,000.) These may be configurable in a future version. However, our parsing poses no problem for languages with alphabets, including computer languages.

Because we tried to be general, we also avoided providing language and domain specific "stop words." Instead, we have attempted to build the ability to automatically recognize words with low discriminatory value into our Bayesian classifier.

Unfortunately, random variation combined with insufficient training examples can cause otherwise uninteresting "noise words" (e.g., "the") to creep into an agent's profile. Therefore, the user may optionally include any noise words into a list of stop words. Stop words are always ignored whenever they are encountered. In particular, they are never used to create profiles nor to classify changes; they are assumed to have zero informative value. Stop words may be any alpha sequence, and there is no practical limit on their number.

4.5.5. The agent's name and description

Users may personalize their agents by providing a name and multi-line description. Both of these fields are free text. While the description is currently saved but ignored by the agent, the name is used to form the subject line of e-mail notifications. Future uses for the description, however, include forming an initial positive training example, as well as allowing users to search for agents using (among other things) the contents of their descriptions. As with all other options, a submit button is provided to cause the agent to store modifications to either of these fields.

4.5.6. Filtering and never-show threshold

The filtering threshold is the minimum score necessary for a change to be considering interesting. There are currently three possible settings; 0.2 (general), 0.5 (average) and 0.8 (strict). The user may be interested in limiting the changes considered to be interesting, or willing to accept a broader definition. For example, agents can act as gateways for several other agents that are then acting as more specific filters. The former will admit many changes, while the latter will be more strict. Most general purpose agents will work well with a setting of 0.5.

4.6. Sharing interesting changes

Our agent supports two modes of collaboration. First, because all communication takes place through a list of e-mail addresses and a Web page, any number of users can simply share the same agent. Simple Unix and Web server access control mechanisms can be used to adjust who can view and modify an agent's activity and configuration. Change notifications can also be sent to distribution lists and news groups. Sharing single agents in this fashion assume a degree of cooperation and coordination among individuals, since agents accept only one vote per change, and modifications to the agent's configuration will affect all users equally.

It is possible, however, for a user to share the information and opinions collected by other user's agents, while allowing them to retain complete control over their own agents. A user does this by cascading agents. An agent's main page functions as a normal Web page with associated URLs, but automatically behaves differently when other agents access them. This

allows users to automatically share information and opinions with others simply by giving them their agent's URL. After this initial transaction, the agents do all the work. Cascaded agents' owners continue to use their agents as before, but now they are also sharing both their lists of interesting sites and their opinions about changes with others. Agents can be cascaded any number of times, allowing the creation of group or organizational "new and interesting" pages.

Because a single agent can monitor an arbitrary number of Web pages, a user can obtain information from any number of other users with equal ease. Normally, obtaining information from many sources can be overwhelming, but cascaded agents only propagate "interesting" information. The user's agent then filters the information again, so only the most relevant information is eventually delivered to the user.

Users of cascaded agents benefit by leveraging the work and judgment of trusted peers. Gathering changes from cascaded agents increases recall, while exploiting others' filtering increases precision. There is little social cost, since no additional work is generated for anyone. Thereby, the perceived costs and benefits favor collaboration.

4.6.1. Receiving changes from other agents

When an agent is given another agent's URL as one of its pages to monitor, the second agent is able to automatically receive all interesting changes detected by the first agent. Agents automatically send structured data rather than HTML if they detect that the requester is another agent. These changes, which are sent by the first agent's associated server using standard HTTP, are also annotated with additional agent-specific information, such as the changes' detection dates, the changes' URLs, and the agent's name.

If an agent detects that a URL points to another agent's main page (currently by matching it's server and path against a set of configurable formats), the agent will request that the other agent directly communicate change information. It does this by slightly modifying the URL to point to a different CGI script before requesting the page from the other agent's server. The "page" returned by the sending agent is a plain-text list of changes and associated unique keys. The requesting agent identifies those keys that it has not seen yet, and returns the associated changes. The sending agent also appends meta-information to each change, such as the agent's name, when it discovered the change, and where it got the change from (either a normal URL or another agent). As we will see later, this meta-information can also be used to determine a change's relevance.

Because we are using a specialized intra-agent communication language to propagate change information, there is no loss in quality between successive agents. This might not be the case if an agent were forced to parse the change from another

agent's human-readable Web page. Interestingly, many of the "non-agent" Web pages that our agent might periodically visit for changes are also generated from databases or other electronic information sources. If the suppliers of these pages were to include an alternative interface that supplied the same information in the form of structured records, rather than HTML, to agents such as ours, automated information collection could be made both more efficient and accurate.

5. SUMMARY

Do-I-Care attempts to solve two post-discovery problems: when should a user revisit a known site for new information, and how does a user share new information with others who may be interested. These post-discovery chores are laborious, suffer from inequitable distribution of costs and benefits, and are not addressed by existing systems.

Because users' agents are under their direct control, the incentive to train them, locate new sources of information, and otherwise augment their capabilities is obvious. Acting entirely in their own best interests, user create valuable repositories of "expert" information -- repositories where the informative value potentially extends beyond the single user to the community at large.

To exploit this information resource, we then gave our agents the ability to communicate what they know with other agents. Because of this, and unlike some collaborative systems, collaborating places no additional burdens on the part of the information providers or users: The normal burdens of transferring and collecting information are handled entirely by the agents. Since agents can easily utilize information discovered by other users' agents, users are motivated to seek out other users with similar interests. There are no corresponding disincentives for these users to share what they have found.

Therefore, by providing a simple technical mechanism, we have found a collaboration mechanism that provides for group effort through individual incentives. We feel this mechanism is generalizable to a large class of problems where users are motivated to train and configure agents to perform useful functions on their behalf, and where the resulting surrogate expert can disseminate its product and expertise at negligible additional cost. In particular, because placing surrogates on the Web facilitates using resource discovery for locating expertise, such a system enables commercial applications where narrow fields of expertise are profitably developed, marketed and disseminated by brokering of surrogate expertise agents.

6. REFERENCES

Ackerman, Mark S. 1993. Answer Garden: A Tool for Growing Organizational Memory. Massachusetts Institute of Technology, Ph.D. Thesis.

Armstrong, Robert, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. 1995. WebWatcher: A Learning Apprentice for the World Wide Web. *Proceedings of the* : 6-12.

Balabanovic, Marko, and Yoav Shoham. 1995. Learning Information Retrieval Agents: Experiments with Automated Web Browsing. *Proceedings of the AAAI Spring Symposium Series on Information Gathering from Heterogeneous, Distributed Environments* : 13-18.

Berger, Peter L., and Thomas Luckmann. 1966. *The Social Construction of Reality: A Treatise in the Sociology of Knowledge*. New York: Anchor.

Bowman, C. M., P. B. Danzig, U. Manber, and M. E. Schwartz. 1994. Scalable Internet Resource Discovery: Research Problems and Approaches. *Communications of the ACM*, 37 (8) : 98-107.

Brown, Carol, Les Gasser, Daniel E. O'Leary, and Alan Sangster. 1995. AI on the WWW: Supply and Demand Agents. *IEEE Expert*, : 50-55.

DeBra, P. M. E., and R. D. J. Post. 1994. Information Retrieval in the World-Wide Web: Making Client-based Searching Feasible. *Computer Networks and ISDN Systems*, 27 (2) : 183-192.

Duda, R., and P. Hart. 1973. *Pattern classification and scene analysis*. New York: John Wiley and Sons.

Koster, Martijn. 1994. ALIWEB - Archie-Like Indexing in the Web. *Computer Networks and ISDN Systems*, 27 (2) : 175-182.

Kuokka, Daniel, and Larry Harada. 1995. Supporting Information Retrieval via Matchmaking. *Proceedings of the AAAI Spring Symposium Series on Information Gathering from Heterogeneous, Distributed Environments* : 111-115.

Maltz, David, and Kate Ehrlich. 1995. Pointing the Way: Active Collaborative Filtering. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95)* : forthcoming.

Pinkerton, Brian. 1994. Finding What People Want: Experieces with the WebCrawler. *Proceedings of the Second International WWW Conference* : http://info.webcrawler.com/bp/WWW94.html.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning*, 1 (1) : 81-106.

Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proceedings of the ACM Conference on Computer-Supported Cooperative Work* : 175-186.

Schwartz, Michael F., Alan Emtage, Brewster Kahle, and B. Clifford Neuman. 1992. A Comparison of Internet Resource Discovery Approaches. *Computing Systems*, 5 (4) : 461-493.

Shardanand, Upendra, and Pattie Maes. 1995. Social Information Filtering. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'95)* : 210-217.

URL-minder. 1996. http://www.netmind.com/URL-minder.