Incremental Clustering for Profile Maintenance in Information Gathering Web Agents

Gabriel L. Somlo and Adele E. Howe
Computer Science Department
Colorado State University
Fort Collins, CO 80523, U.S.A.
{somlo, howe}@cs.colostate.edu

ABSTRACT

User profiles are the central component of most personalized Web information agents. They consist of a set of models representing the various topics of interest to the user. Often the agent learns the user's preferences from examples of documents deemed relevant to the user. The topic of the document can either be supplied by the user (active modeling), or it must be guessed by the agent (passive modeling), which is more convenient but is expected to diminish the agent's accuracy. We present an empirical study assessing the trade-offs in passive versus active document classification. We compare a manual profile maintenance technique in which the user supplies the document topic, and two incremental clustering methods (greedy and the doubling algorithm) for automated maintenance of the user profile components. The study is performed using our SurfAgent, a testbed information gathering Web agent. Our evaluation methodology exploits the strong parallel between Web information agents and text filtering; we use text filtering benchmarks from the information retrieval community (TREC disk #5) to simulate user behavior and thus speed up data collection, exert additional experimental control and improve the objectivity of our results.

Keywords

information agents, user modeling, adaptation and learning

1. INTRODUCTION

In recent years, the amount of information available on the Web has been increasing dramatically. The problem users are faced with today is no longer the lack of useful information, but that of finding information pertinent to their personal needs.

Many tools have been implemented to address this problem. From the early days of the Web, search engines have proliferated, but each cannot keep up with the tremendous growth of the Web [13]. Meta-search engines, such as MetaCrawler [19, 20], SavvySearch [8, 9, 10], and NECI [12], propose to overcome the Web coverage problem by combining the indexing power of multiple stand-alone search engines. However, because they leverage the capabilities of many search engines, they tend to generalize the search task: limiting the access to search-engine-specific advanced search capabilities and, perhaps, introducing even more noise into the return results. To compensate for this trend toward the least common denominator, many search and meta-search engines support some customization (e.g., types of search, return amount, and search engine selection).

An alternative to the generally available search engines is a personal Web information agent (a *helper*). Agents, such as Letizia [15], WebWatcher [1, 11], Fab [3, 4], and WebMate [7], can be tailored, over time, to help their users find information that matches their own interests. These agents generally comply to the architecture presented in Figure 1. The agent intercedes between the user and their Web access, learning a model of user requests to be used in modifying requests and/or filtering results.

Each personal helper agent follows the same basic architecture but offers somewhat different help to its users. Letizia pre-fetches Web pages for its user (anticipating the user's interest) by exploring the links in the Web page currently being viewed. Similarly, WebWatcher recommends links, in general, that are likely to lead to interesting documents; it ranks a fixed collection of links, available at the time, in decreasing order of relevance with respect to the user profile. Fab builds a list of likely to be relevant documents through best-first search; documents that pass the filtering phase are then included in a list of recommended links. Thus, Fab separates the generation of the document stream from the subsequent filtering. Finally, WebMate uses greedy incremental clustering to build a user profile, which is used to compile a personal newspaper composed of articles found on a list of well-known news sources.

In this paper, we study a critical component of personal information agents: modeling the users' information categories of interest. The user model summarizes and generalizes each user's interests, in service of determining when new information is likely also to be of interest. Most users have diverse information interests encompassing their work (e.g., "information gathering agents"), their hobbies (e.g., "Italian cuisine"), current issues (e.g., "presidential debates"), etc. Generalizing across all of them is clearly not effective. Instead, topics are classified into separate categories, and

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.

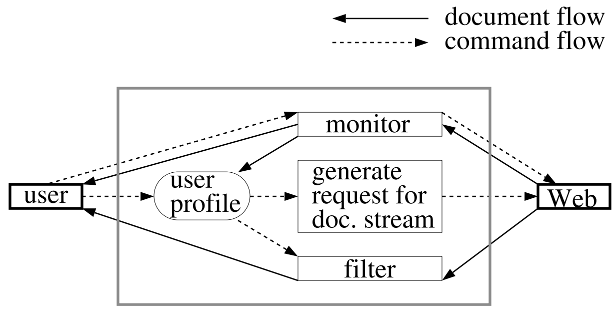


Figure 1: Architecture of a Web information helper agent

the categories form the generalizations.

In Section 2, we examine the architecture of a typical information helper agent and point out the strong connection between the science of building such agents and text filtering in information retrieval. We describe the clustering problem for automated maintenance of user profiles, and a set of topic clustering algorithms in Section 3. We compare the efficacy of three basic clustering algorithms (and different parameter settings) in an experiment in Section 4, and discuss trade-offs identified in the results in Section 5.

2. TEXT FILTERING AND PERSONAL WEB INFORMATION AGENTS

Text filtering is a relatively recent area of information retrieval (IR). Classic IR methods rank documents in a fixed collection by relevance to a user query. In contrast, text filtering makes binary decisions of whether or not to disseminate items from a continuously incoming stream of documents [5]; the decision is made based on whether a document adequately matches a model of categories deemed to be of interest. In text filtering, models of categories are constructed from vectors of terms in disseminated documents; the models are updated frequently, making the categories better match the user's information requirements over time. Thus, the basis for decision making (queries for classic IR and categories for text filtering) has a much longer lifespan in text filtering than in classic IR systems.

There are strong similarities between helper agents (mentioned in Section 1) and a text filtering system. Both types of systems examine an incoming document stream; agents often generate such a stream for themselves by crawling the Web, querying a search engine, or by simply extracting all links from the Web page the user is looking at. Both types of systems maintain a model of queries (in the Web agent community, this is also referred to as the *user profile*). Finally, both systems make binary decisions about the documents in the incoming stream: whether to recommend them to the user based on their similarity to the model of queries (categories or user profile).

In text filtering systems, separate categories are maintained for each distinct information need or topic of interest, which tend to be fairly narrow. Every time a document is successfully disseminated, the user has the opportunity to update the specific category for which it was found relevant. This is done using a technique known as *relevance feedback* [17], in which topic vectors (category models) are adjusted to match a user's determination of relevance. Relevance Feedback modifies TF-IDF (Term Frequency-Inverse Document Frequency) vectors, which encode weights for each known query/document term. When the user gives a positive rating to a document, the document's TF-IDF vector is normalized and weighted by a document weight, w; this vector is then added to the topic's vector (called a *query vector*). The document weight is a parameter of most algorithms and dictates how much credence to lend to a new document over the query vector compiled from perhaps many documents.

Users of Web helper agents, on the other hand, need a much wider range of topics of interest. One of the major design issues is how to construct, maintain and use multiple different topics/categories that are contained in the user profile. The most straightforward option, and the one closest to the text filtering paradigm, is to give the user complete control over the creation and updating of these categories. Every time an interesting document is found, it is the responsibility of the user to identify the applicable topic of interest on which to perform relevance feedback. If users would diligently maintain their user profile, the agent's recommendations could achieve a very high accuracy.

Unfortunately, the maintenance of a user profile containing explicit categories can become rather burdensome, e.g., requiring intervention after reading every interesting Web page. Therefore, in a departure from the methods employed for text filtering, authors of personal information agents usually choose to automate the maintenance of the user profile. Under this paradigm, there is no longer a strict mapping between the components of the user profile and a user's designated topics of interest. For example, in WebMate, the user is simply required to indicate when a document is interesting, regardless of which topic of interest it is relevant to. An incremental clustering algorithm is used to automatically select the component of the user profile (pseudo-topic) that will receive the update.

3. INCREMENTAL CLUSTERING ALGORITHMS

The clustering problem consists of partitioning n points in space into k clusters so as to minimize the maximum cluster size. The clustering method most frequently used for IR is *hierarchical agglomerative clustering* (HAC) [16]. This method initially assigns the n original points to n clusters, and then repeatedly merges pairs of clusters, using the document TF-IDF vectors, until only k clusters are left. The boundaries of the cluster space generalize the documents that have been seen into a model of the categories.

Unfortunately, straightforward clustering is not feasible for maintenance of Web agent user profiles. It is impractical to store all the relevant documents and periodically apply a clustering algorithm to build the components of the profile. Instead, only the cluster representations should be kept in the profile, and document representations should be discarded immediately after the cluster representations have been updated. Algorithms for solving this new problem of *incremental clustering* have been studied in [6]. Essentially, the problem of incremental clustering consists of maintaining k clusters such that when a new point from an incoming sequence is presented, it is either assigned to one of the existing k clusters, or a new cluster is created from it while at least two other existing clusters are collapsed into one.

In our study, we compare three techniques of categorizing new relevant documents as user profile components: the user-driven technique of explicitly providing the topic of interest associated with the document as in text filtering; the straightforward, greedy, incremental clustering method used in current Web helper agents (e.g., WebMate); and an advanced, incremental clustering technique known as the *doubling algorithm* [6]. The remainder of this section discusses first the components of the implementation shared by the three algorithms (i.e., the representation of clusters and the basic concepts underlying their construction and usage) and then presents the specifics of the three algorithms.

3.1 Cluster Representation and Construction Basics

In information retrieval, documents are represented as TF-IDF vectors. Each document d is converted into a vector in a multi-dimensional space, with one dimension for each term t in the vocabulary. The components of such a vector are computed using the following well-known formula [18]:

$$TFIDF_{d,t} = TF_{d,t} \cdot \log \frac{D}{DF_t}$$

where $TF_{d,t}$ is the number of occurrences of term t in document d (*term frequency*), and the logarithm is referred to as the *inverse document frequency* of term t. D is the total number of available documents, and DF_t is the number of documents containing term t.

For TF-IDF computation, D and DF values assume the existence of a fixed corpus, where all possible documents are known a priori. In our Web application, we cannot assume random access to all documents. Thus, we adopt the model of text filtering and instead treat the Web as a source of incoming documents. D and DF are therefore being approximated: every time a new incoming document is processed, we increment D and update the entries into DF that correspond to the terms contained in the new document. The similarity between two documents is computed using the cosine measure, i.e., the normalized dot product between their respective TF-IDF vectors.

Each cluster in our user profile consists of a normalized TF-IDF query vector corresponding to a model of the documents that should be relevant to the cluster's topic. Query vectors are learned using relevance feedback (see Section 2). The only departure from the IR version of the technique is the previously mentioned computation of D and DF.

A document is disseminated to a user when its TF-IDF vector is deemed similar to the query vector. Exactly how similar the vectors must be is not described for the other helper agents. In our formulation, as in text filtering applications, associated with each query vector is a dissemination threshold. Documents are considered to be relevant to the cluster's topic if their similarity with the cluster's vector exceeds the associated dissemination threshold. Like the query vector contents, the dissemination threshold is also adapted [5, 21, 22]. We can imagine the tip of the query vector to be at the center of a "bubble", with a radius given by the dissemination threshold. If the new relevant document supplied by the user as feedback is inside this bubble, we can raise the threshold and "shrink" the bubble, and thus increase the filtering precision. If the feedback document was outside the bubble, it means that the user would like more recall, and we should "inflate" the bubble and lower the

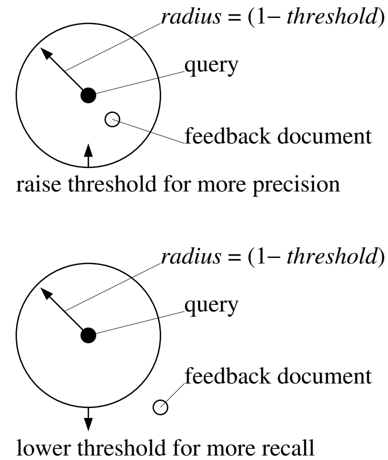


Figure 2: Learning the dissemination threshold

AddDocToCluster($docvec, cluster$)
$s =$ similarity($cluster.vector, docvec$);
normalize($cluster.vector$);
normalize($docvec$);
$cluster.vector \mathrel{+}= w \cdot docvec$;
normalize($cluster.vector$);
$cluster.threshold \mathrel{+}= \alpha \cdot (s - cluster.threshold)$;
$cluster.npoints \mathrel{+}= 1$;

Figure 3: Adding a document to a cluster

dissemination threshold. This process is illustrated graphically in Figure 2. In essence, the dissemination threshold tracks the similarity between the original query vector and the feedback document, with a learning rate α. Our implementation, presented in Figure 3, is a simplified version of the method used in [21, 22], which required maintenance of a large document history for periodic recomputations of the dissemination threshold.

Under incremental clustering, clusters periodically need to be merged to maintain the established count. The merging acknowledges that two clusters may have moved close enough together to be better captured as a single generalization. Established clustering algorithms usually assume that clusters are represented by their center of mass and a diameter. When clusters are merged, the new center of mass is computed from all points now contained in the resulting cluster, and the diameter is set to the distance between the two points in the cluster that are the farthest apart. In order to adapt this principle to TF-IDF vectors, when two clusters are merged in our implementation, the query vector becomes the sum of the two original query vectors, weighted by their mass (the number of points contained in each original cluster). The same is true for the dissemination threshold, which is set to the weighted sum of the dissemination thresholds of the original clusters. This operation is described in Figure 4. Note that if the second cluster involved in the merging operation consists of a single point, we consider this to be a document addition to the first cluster. The merging thus defaults to the AddDocToCluster procedure from Figure 3.

Alternatively, when a feedback document represents a sig-

```
MergeClusters(cl1, cl2)
    if(cl2.npoints) is 1 then
      AddDocToCluster(cl2.vector, cl1);
      return(cl1);
    else
      normalize(cl1.vector);
      normalize(cl2.vector);
      newcl.vector =  cl1.npoints · cl1.vector+
                      cl2.npoints · cl2.vector;
      normalize(newcl.vector);
      newcl.threshold =  (cl1.npoints · cl1.threshold+
                          cl2.npoints · cl2.threshold)/
                         (cl1.npoints + cl2.npoints);
      newcl.npoints = cl1.npoints + cl2.npoints;
      return(newcl);
```

Figure 4: Merging two clusters

```
FeedbackXPL(doc, topic)
    AddDocToCluster(doc.vector, clusters[topic]);
```

Figure 5: Agent feedback with explicit cluster assignment by the user

nificant departure from the existing clusters, a new cluster must be created for it. The query vector is initially set to the normalized document vector, and the dissemination threshold is initialized to 0.5.

We considered a simple geometric-based alternative to the dissemination thresholds: compute a new cluster center and radius such that all points contained by the cluster would be contained in the resulting sphere. Dissemination would be decided based on whether the candidate document was inside the sphere. We decided against using this approach, because it only addresses recall, ignoring precision: the more documents added to a cluster, the larger it becomes.

3.2 Explicit Cluster Assignment

Users know their topics of interest, making them the best source of evidence of cluster membership. Thus, under explicit cluster assignment, the user tells the agent to which topic to add the document. The agent maintains one cluster for each topic and adds the weighted document vector to the cluster corresponding to the supplied topic. Our implementation of this method is shown in Figure 5. Although we do not view this method as ultimately viable for Web helper agents, we include it to serve as the benchmark for the best performance that could be achieved.

3.3 Greedy Incremental Clustering

Users are not required to define and maintain the models of their topics of interest. When feedback is offered (the user simply indicates interest in the document), the agent is presented with the document and must determine for itself in which pseudo-topic in the profile to include it.

The greedy algorithm maintains a fixed number, k, of topic clusters. The algorithm initially assigns each of the first k documents offered as feedback to its own cluster. Once k clusters have been formed, each new document is treated as the $(k+1)$st cluster, and the algorithm then pro-

```
FeedbackGRD(doc)
    clusters[k + 1] = newcluster(doc);
    (i, j) = closest two clusters, with i < j;
    newcl = MergeClusters(clusters[i], clusters[j]);
    delete(clusters[i]);
    delete(clusters[j]);
    insert(newcl);
```

Figure 6: Agent feedback using greedy incremental clustering

ceeds to merge the closest two clusters in order to bring the number of clusters back to k. Thus a new document may be placed in an existing cluster or may form its own cluster forcing existing clusters to merge.

The pseudocode for this algorithm is in Figure 6. Greedy Incremental Clustering is the simplest of the two incremental clustering algorithms in our study. It captures the current practice in personal Web agents (e.g., WebMate).

3.4 The Doubling Algorithm

As with greedy clustering, initially, each feedback document, up to the kth, is placed in a separate cluster. However, merging allows mutiple clusters to be merged, such that the total number of clusters never exceeds k, but need not equal it either. It instead emphasizes the distance between clusters and determines which clusters to merge based on a distance threshold, d.

When the number of clusters first exceeds k, d is initialized to twice the distance between the closest two clusters. Then a merging stage is executed in which groups of clusters *close* to each other are joined in order to reduce the total number of clusters below k. Close is defined such that the distance between clusters cannot exceed d.

During feedback, a document is added to an existing cluster if the distance between their TF-IDF vectors is less than $2d$, or else it is placed in a new cluster. After each feedback document is inserted, the algorithm checks whether the number of clusters exceeds k, and, if so, executes another merging stage. Before each merging stage, d is doubled, hence the name of the algorithm. The pseudocode for this procedure is presented in Figure 7. Because the threshold is twice the known minimum, several pairs of clusters may be merged during the process.

The doubling algorithm was originally developed for points in a metric space, and therefore it assumes that the distance between two points can take values anywhere in the interval $[0, \infty)$. However, for IR applications such as this, we compute similarity between two TF-IDF vectors using the cosine measure, which ranges from 0 for dissimilar to 1 for a perfect match. Therefore, for our application, we convert similarity into distance using the following formula:

$$\text{dist}(vector1, vector2) = \frac{1}{\text{similarity}(vector1, vector2)} - 1$$

which maps a similarity of 1 to a distance of 0, and a similarity of 0 to a distance of ∞.

This algorithm has been theoretically proven [6] to perform better than the greedy algorithm. The metric used was the *performance ratio* of the incremental algorithm, which is the ratio of its maximum cluster diameter to that of the

```
FeedbackDBL(doc)
    i = closest cluster to doc.vector;
    if dist(clusters[i].vector, doc.vector) ≤ 2d then
      AddDocToCluster(doc.vector, clusters[i]);
    else
      insert newcluster(doc);
    while number of clusters > k do merging stage :
      d = 2d;
      build graph G: for all pairs (i, j) do
        if dist(clusters[i].vector, clusters[j].vector) < d
        then connect i and j;
      repeat
        pick arbitrary node n in G;
        foreach neighbor m of n do :
            MergeClusters(n, m);
            disconnect node m from G;
        disconnect node n from G;
      until empty(G);
```

Figure 7: Agent feedback using doubling incremental clustering

optimal clustering for a given set of points. The doubling algorithm is proven to have a performance ratio of 8, whereas the greedy algorithm has a performance ratio of $(2k - 1)$, which degrades with the number of clusters. Thus, we include it in the set because it would appear to offer a superior alternative to current practice.

4. EXPERIMENT

Typically, the performance of existing Web agents has been evaluated with user studies. While we think that users are the ultimate arbiters of the quality of a Web agent, such studies, in practice, can be difficult to run properly and impractical for testing all design decisions. In particular, users do not wish to waste their time unless the system provides some servce, and they may be biased by exogenous factors (e.g., the user interface) to the study. Thus, we advocate running off-line studies with benchmark data sets to test intermediate design decisions.

The purpose of our study is to determine whether the choice of clustering algorithm affects performance and if it does, which algorithm, under what parameter settings, provides the best design trade-off. To motivate our study, we exploit the strong parallel between Web information agents and text filtering in the design of our experiment. We use a benchmark from TREC[1] (Text REtrieval Conference, a well established and supported regular comparison of IR systems) in order to simulate the behavior of Web helper agent users. This benchmark (TREC disk #5) consists of 258,213 news articles. There are a total of 450 different topics that have been used at various TREC conferences. For each of these topics, a subset of the documents has been rated as *relevant* or *non-relevant*. The disk acts as a rather curious user who is interested in these 450 topics and who would like a Web helper agent to disseminate articles relevant to these topics from the incoming document stream.

[1]TREC benchmark disks are publically available and can be ordered from the conference homepage at `http://trec.nist.gov/`

As the basis for our studies of Web information helper agents, we have developed a testbed Web agent, called *SurfAgent* [2], which follows the basic architecture described in Section 2 and is designed to expedite plug-and-play of alternative algorithms, front-ends, and service tasks. For this study, we have implemented the three algorithms and set-up SurfAgent so that they each can be tested with different parameter settings. In addition, we have redirected the I/O so that the TREC data can be used instead of standard Web and user interaction.

The experiment proceeds as follows: Clusters are initialized as described for each algorithm. For each new incoming document, SurfAgent makes a relevance prediction based on the current user profile: the similarity between the document vector and each topic vector is compared to the respective topic's dissemination threshold. For the explicit clustering case, the prediction is made for each existing profile topic. For the incremental algorithms, only a relevant/non-relevant prediction is made: if the document passes at least one of the internal profile topic dissemination thresholds, it is considered relevant.

After SurfAgent makes its prediction, the user has the opportunity to give feedback. We assume that users only bother to give feedback on relevant documents. This is simulated using the relevance ratings that are associated with the benchmark disk. For the explicit clustering case, we add each document to all topics for which it is known a priori to be relevant; note: a document can be designated to be relevant to more than one of the 450 topics. For the incremental algorithms, the document is presented as a positive feedback example if it was relevant under at least one a priori topic.

The following quantities are recorded during these experiments:

- RF – Relevant Found – number of relevant documents correctly identified as relevant by the agent;
- RM – Relevant Missed – number of relevant documents incorrectly identified as non-relevant;
- NF – Non-relevant Found – number of non-relevant documents incorrectly identified as relevant;

In the case of explicit clustering, we imposed the additional condition that relevance predictions have to match a priori judgments *in the correct category*. In other words, if the document is known to be relevant to one topic, and the agent finds it relevant, but under a different topic, we increment both RM and NF, but not RF.

Some of the documents contained in the benchmark have not been rated for any of the available topics (about 168,000). We only use these documents to improve our estimates of D and DF (see Subsection 3.1), but do not use them in the computation of RF, RM, and NF.

We ran a factorial experiment, with the following independent variables:

- Algorithm – the three algorithms presented earlier in the paper:
  - XPL: explicit cluster assignment by the user;
  - GRD: automatic incremental clustering with greedy assignment strategy;
  - DBL: automatic incremental clustering using the doubling algorithm;

- α – the dissemination threshold learning rate, with allowed values in the interval $[0, 1]$. The values used in our tests were 0.1, 0.5, and 0.9.
- w – the weighting factor by which document vectors are scaled before being added to a cluster. Sub-unit values of w are used to assign more weight to the original cluster vector, whereas values greater than one are used to favor the new document vector. We used the values 0.25, 1, and 4 for w during our tests.
- k – the maximum number of allowed clusters, applicable only to GRD and DBL. The values used were 50, 100, and 150.

For each combination of parameters, we recorded RF, RM, and NF, and used them to compute values for recall and precision, according to the following formulas:

$$recall = \frac{RF}{RF + RM}$$

$$precision = \frac{RF}{RF + NF}$$

The value of *recall* reflects how many relevant documents were correctly disseminated by the agent. The value of *precision* reflects how many of the disseminated documents were actually relevant.

Additionally, in order to assess the overall performance of an algorithm in terms of both recall and precision, we used the Lewis and Gale F-measure[2] [14]:

$$LGF = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Our expectations, before running the tests, were that XPL would be the overall best performing algorithm, due to the tremendous extra information it receives from the user. Also, we expected DBL to outperform GRD, and, for both GRD and DBL, combinations with larger k to outperform combinations with lower k.

5. RESULTS

We found statistically significant differences in performance between the algorithms on the three metrics. One-way Analyses of Variance (ANOVAs) on algorithm for the metrics produced $F = 43.15, F = 12.5, F = 30.7, p < .0001$ for recall, precision, and combination, respectively. Consequently, we present the results for each of the parameter settings on each of the performance metrics in separate tables for the algorithms. For each algorithm, we statistically analyzed the data, using a ANOVAs to determine whether the algorithm's performance was sensitive to each of the parameters.

The results for XPL are presented in Table 1. One-way ANOVAs on each of the parameters and performance metrics showed that XPL's performance is insensitive to w ($p > .28$ for all three measures). Recall and precision are sensitive to α ($F = 10.7, p < .01$ and $F = 132.1, p < .0001$ respectively), but their combination is marginally dependent ($F = 4.64, p < .06$). Increasing the value of α improves recall but degrades precision. The effect on precision is stronger however, and therefore the best overall results (LGF) are found for small values of α. Although not a significant difference, small values for the document weight w somewhat improve recall. Overall, as measured by LGF, the best result for this algorithm is obtained for $\alpha = 0.1$ and $w = 0.25$.

α	w	*recall*	*prec.*	LGF
0.10	0.25	0.4063	0.4647	0.4336
	1.00	0.3481	0.4847	0.4052
	4.00	0.2992	0.4520	0.3601
0.50	0.25	0.4787	0.3256	0.3876
	1.00	0.4521	0.3266	0.3793
	4.00	0.3977	0.2905	0.3358
0.90	0.25	0.5086	0.2566	0.3411
	1.00	0.5075	0.2505	0.3354
	4.00	0.4853	0.2310	0.3130

Table 1: Results for explicit cluster assignment

As with XPL, GRD's performance on all three metrics (see Table 2) depends on α ($F = 51.9, F = 22.0, F = 30.0, p < .0001$ for recall, precision, and combination); α is correlated with recall and combination and inversely correlated with precision. Although not statistically significant ($F = 2.63, p < .09$), overall, better performance is obtained for $w \leq 1$. k has even less effect on performance ($F = 0.12, p < .89$, $F = .32, p < .73$, and $F = 0.08, p < .92$ for recall, precision, and combination), but larger ks seem to help slightly. The best overall result, as measured by LGF was obtained for $\alpha = 0.9$, $w = 1$, and $k = 150$.

As with the other two algorithms, large values for α translated to better recall and worsened precision for DBL (see Table 3). Overall, small values of α are better. Small values of w lead to better recall, and precision is best for $w = 1$. Statistical analysis found DBL's performance sensitive to α ($p < .0001$ for all three metrics) and insensitive to the other parameters ($p > .45$ for all) except for effect of w on LGF ($F = 6.67, p < .005$). The overall best result for the doubling algorithm was obtained for $\alpha = 0.1$, $w = 0.25$, and $k = 50$.

The extreme insensitivity of DBL to k ($p > .94$) means that values even lower than 50 could be appropriate. This is impressive, considering that the total number of real categories in our test suite was 450.

The best recall was obtained by DBL (max = 0.627, mean = 0.493). The best precision was obtained, surprisingly, with GRD (max = 0.530, mean = 0.356). The best LGF was, as we expected, obtained by XPL (max = 0.434, mean = 0.366). Also, with respect to the LGF metric, DBL outperforms GRD for every parameter combination we tried.

Our expectation that increasing k will improve performance was observed only for GRD, and even then was not statistically significant. Of the parameters, only α significantly affected performance.

Our experiment leverages far more examples than a personalized Web helper is ever likely to encounter (although far less than what a community/shared Web agent will). Consequently, we examined the learning curves for precision, recall, and LGF for the overall best parameter settings (as measured by LGF) for XPL, GRD, and DBL in Figure 8. All three algorithms appear to require close to 10,000 feed-

[2] F also is the statistic for the Analysis of Variance statistical test. Because we will be using that test to analyze our data, we will call this measure LGF instead, to avoid confusion with the F statistic.

α	w	k	*recall*	*prec.*	*LGF*
0.10	0.25	50	0.2631	0.2894	0.2756
		100	0.2144	0.3441	0.2642
		150	0.1902	0.3796	0.2534
	1.00	50	0.1564	0.5077	0.2391
		100	0.1563	0.5057	0.2388
		150	0.1600	0.5186	0.2446
	4.00	50	0.1199	0.5082	0.1941
		100	0.1272	0.5126	0.2038
		150	0.1274	0.5301	0.2054
0.50	0.25	50	0.3307	0.2591	0.2906
		100	0.2910	0.3034	0.2971
		150	0.2844	0.3324	0.3065
	1.00	50	0.2540	0.3731	0.3023
		100	0.2544	0.3767	0.3037
		150	0.2588	0.3912	0.3115
	4.00	50	0.2260	0.3332	0.2693
		100	0.2263	0.3450	0.2733
		150	0.2239	0.3683	0.2785
0.90	0.25	50	0.3963	0.2285	0.2899
		100	0.3647	0.2585	0.3026
		150	0.3533	0.2709	0.3067
	1.00	50	0.3376	0.2845	0.3088
		100	0.3358	0.2967	0.3151
		150	0.3325	0.3031	0.3171
	4.00	50	0.3357	0.2531	0.2886
		100	0.3251	0.2628	0.2907
		150	0.3201	0.2800	0.2987

Table 2: Results for the greedy algorithm

α	w	k	*recall*	*prec.*	*LGF*
0.10	0.25	50	0.4343	0.3003	0.3551
		100	0.4321	0.2912	0.3479
		150	0.4227	0.2947	0.3472
	1.00	50	0.3738	0.3192	0.3444
		100	0.3689	0.3396	0.3536
		150	0.3580	0.3433	0.3505
	4.00	50	0.3137	0.3138	0.3138
		100	0.3133	0.3186	0.3159
		150	0.2920	0.3354	0.3122
0.50	0.25	50	0.5217	0.2447	0.3332
		100	0.5278	0.2382	0.3283
		150	0.5187	0.2422	0.3302
	1.00	50	0.4930	0.2364	0.3195
		100	0.5124	0.2431	0.3298
		150	0.4895	0.2539	0.3343
	4.00	50	0.4512	0.2231	0.2986
		100	0.4533	0.2260	0.3016
		150	0.4489	0.2327	0.3065
0.90	0.25	50	0.6199	0.2054	0.3086
		100	0.6268	0.1987	0.3018
		150	0.6062	0.2037	0.3050
	1.00	50	0.6179	0.1929	0.2941
		100	0.6452	0.1972	0.3020
		150	0.6219	0.2051	0.3085
	4.00	50	0.6122	0.1809	0.2793
		100	0.6108	0.1836	0.2823
		150	0.6113	0.1868	0.2862

Table 3: Results for the doubling algorithm

back operations before they stabilize. This is most likely due to the relatively large number of topics (450) that the agent has to track. In real life situations, we do not expect human users to have more than 20 different topics of interest, which may allow the algorithms to stabilize much faster. Prior to 5,000 documents, the overall performance is comparable, although the precision and recall results are somewhat distinct.

In assessing the design trade-offs, we did not collect computation times or study the quality/quantity of the feedback that users would supply. We knew the differences in computation time to be negligible.

6. CONCLUSIONS

Our study empirically confirms the superiority of the doubling algorithm over greedy incremental clustering for overall filtering performance. Further experiments will determine how much we can still decrease the number of allowed clusters k before observing a significant loss of performance.

Both incremental algorithms are outperformed by XPL when overall performance is measured. However, if recall is the main concern, DBL is the best algorithm. When high precision is desired (i.e., parameters are tuned for precision), GRD outperforms the other algorithms.

While it appears that all tested algorithms take a large amount of training before they stabilize, we think that this is due to the large number of topics being tracked, and that further experiments will prove that when fewer topics are tracked, less training is needed by the algorithms.

We think that for intermediate design decisions, a factorial experiment is much more appropriate than a user study. It is cheaper, faster, and the results are more objective and conclusive. When we have tested all of the key design decisions in SurfAgent, we will confirm this hypothesis with a full user study of its resulting functionality.

7. ACKNOWLEDGMENTS

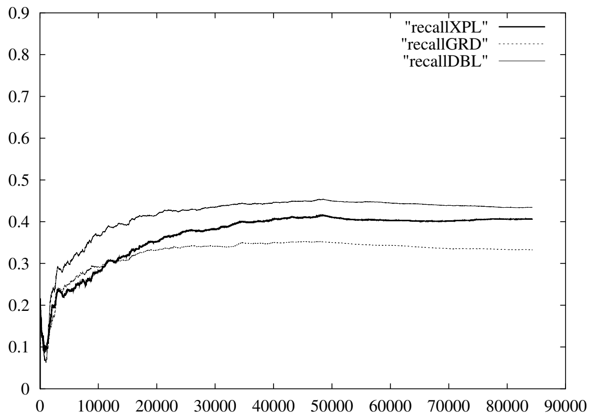
This research was supported in part by National Science Foundation Career Award IRI-9624058. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation herein. We also wish to thank our reviewers for their helpful comments.

8. REFERENCES

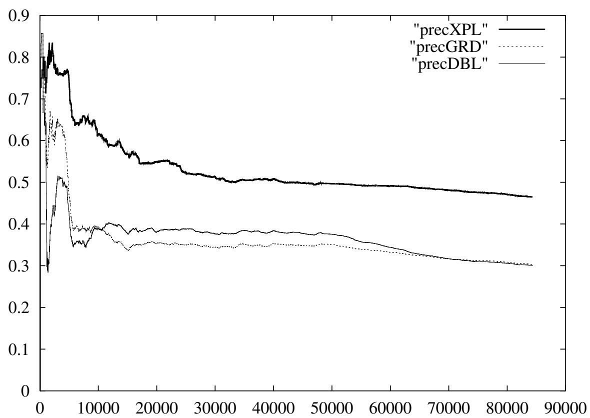
[1] Armstrong, R., Freitag, D., Joachims, T., Mitchell, T. WebWatcher: A Learning Apprentice for the World Wide Web *Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Resources*, Stanford, CA, 1995.

[2] Somlo, G.L., Howe, A.E. Agent-Assisted Internet Browsing. *Working Notes of AAAI-99 Workshop on Intelligent Information Systems*, Orlando, FL, July 1999.

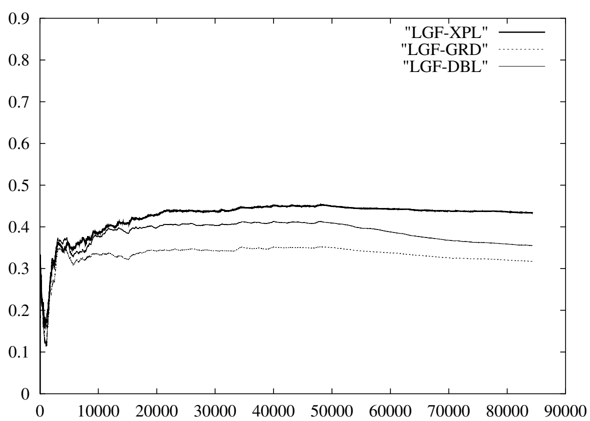
[3] Balabanović, M., Shoham, Y. Learning Information Retrieval Agents: Experiments with Automated Web Browsing. *Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Resources*, Stanford, CA, 1995.



a) recall



b) precision



c) *LGF* metric

Figure 8: Learning curves for best instances of XPL, GRD, and DBL as measured by *LGF*

[4] Balabanović, M. An Adaptive Web Page Recommendation Service. *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, 1997.

[5] Callan, J. Learning While Filtering Documents *Proceedings of the 21st International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia, 1998.

[6] Charikar, M., Chekuri, C, Feder, T., Motwani, R. Incremental Clustering and Dynamic Information Retrieval *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.

[7] Chen, L., Sycara, K. WebMate: A Personal Agent for Browsing and Searching. *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, MN, 1998.

[8] Dreilinger, D., Howe, A.E. An Information Gathering Agent for Querying Web Search Engines. Colorado State University, Computer Science Department, TR-CS-96-111, 1996.

[9] Dreilinger, D., Howe, A.E. Experiences with Selecting Search Engines using Meta-Search. *ACM Transactions on Information Systems*, 15(3):195-222, 1997.

[10] Howe, A.E., Dreilinger, D. SavvySearch: A Meta-Search Engine that Learns Which Search Engines to Query. *AI Magazine*, 18(2):19-25, 1997.

[11] Joachims, T., Freitag, D., Mitchell, T. WebWatcher: A Tour Guide for the World Wide Web. *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Japan, 1997.

[12] Lawrence, S., Giles, C.L. Context and Page Analysis for Improved Web Search. *IEEE Internet Computing*, 2(4):38-46, 1998.

[13] Lawrence, S., Giles, C.L. Accessibility of Information on the Web. *Nature*, 400:107-109, 1999.

[14] Lewis, D.D., Gale, W.A. A Sequential Algorithm for Training Text Classifiers *Proceedings of the 17th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, 1994.

[15] Lieberman, H. Letizia: An Agent That Assists Web Browsing. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

[16] Rasmussen, E. Clustering Algorithms. *Frakes, W., Baeza-Yates, R. (eds.), Information Retrieval: Data Structures and Algorithms* Prentice-Hall, 1992.

[17] Rocchio, J.J. Relevance Feedback in Information Retrieval *Salton, G. (ed.), The SMART Retrieval System: Experiments in Automatic Document Processing* Prentice-Hall, 1971.

[18] Salton, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer* Addison-Wesley, 1988.

[19] Selberg, E., Etzioni, O. Multi-Service Search and Comparison Using the MetaCrawler. *Proceedings of the Fourth International World Wide Web Conference*, Boston, MA, 1995.

[20] Selberg, E., Etzioni, O. The MetaCrawler Architecture for Resource Aggregation on the Web. *IEEE Expert*, 12(1):8-14, 1997.

[21] Zhai, C., Jansen, P., Stoica, E., Grot, N., Evans, D.A. Threshold Calibration in CLARIT Adaptive Filtering *Proceedings of the Seventh Text REtrieval Conference (TREC-7)*, Gaithersburg, MD, 1998.

[22] Zhai, C., Jansen, P., Roma, N., Stoica, E., Evans, D.A. Optimization in CLARIT TREC-8 Adaptive Filtering *Proceedings of the Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, MD, 1999.