Collaborative Filtering with Maximum Entropy

Dmitry Pavlov, *Yahoo*

Eren Manavoglu and C. Lee Giles, *Pennsylvania State University*

David M. Pennock, *Yahoo Research Labs*

As users navigate through online document collections on high-volume Web servers, they depend on good recommendations. The authors present a novel maximum-entropy algorithm for generating accurate recommendations and a data-clustering approach for speeding up model training.

Recommender systems attempt to automate the process of "word of mouth" recommendations within a community. Typical application environments such as online shops and search engines have many dynamic aspects: users come and go, users' preferences and goals change, items are added and removed, and the user navigation itself shifts. Recommendation domains are also often high dimensional and sparse, with tens or hundreds of thousands of items, few of which are known to any particular user.

Consider, for instance, the problem of generating recommendations in ResearchIndex (also known as CiteSeer, http://citeseer.ist.psu.edu)—an online digital library of computer science papers receiving thousands of user accesses per hour. The site automatically locates computer science papers posted on the Web, indexes their full text, enables browsing via a literature citation graph, and extracts the citations' context, among other services.[1] We compared several probabilistic model-based collaborative recommenders to the similarity-based recommenders currently available in ResearchIndex and to the standard, often-used correlation method (our baseline). We chose to work with the ResearchIndex data because it's a rich data source with properties typical of many recommendation application areas.[2] Our methods should work with little adaptation in other domains. This article presents a novel maximum entropy, or maxent, approach for generating online recommendations. Maxent enables fast model querying, is relatively compact, and generally performs as well as or better than its competitors in terms of accuracy. We also suggest a document-clustering approach that helps speed up training of the model.

Calculating recommendations

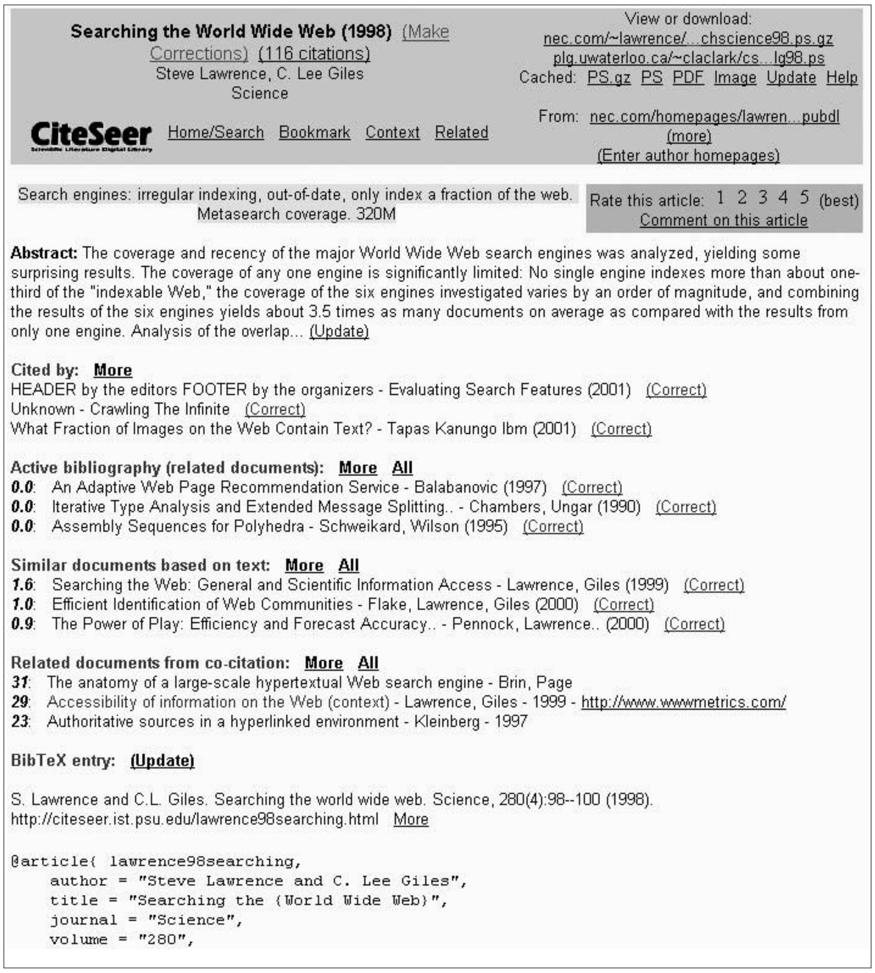
There are two conceptually different ways of making recommendations. *Content-filtering* methods recommend solely based on the features of a document D (for example, showing documents written by the same authors or documents textually similar to D). *Collaborative-filtering* methods[3] work by assessing the similarities among users on the basis of their overlap in document requests, then recommending to a given user documents that like-minded users accessed previously. Common measures of similarity between users include correlation, mean squared error, vector similarity, and Bayesian similarity measures. Other approaches to making recommendations include statistical machine-learning techniques such as Bayesian networks, dependency networks, singular-value decomposition, and latent class models.

Figure 1 shows a typical screen shot of a document details page in ResearchIndex. The page shows the paper's title and authors, download options, and a number of recommenders predicting documents of possible interest to the user on the basis of the current document's features. The archive contains more than 470,000 documents, including the full text of each, citation links between documents, and a wealth of user access data. With so many documents and only eight accesses per user on average, the user-document data matrix is exceedingly sparse and thus challenging to model. ResearchIndex uses several *similarity-based* recommenders, most of which are content-based and user-independent. On the other hand, most collaborative-filtering algorithms are user-specific but context- and order-independent: that is, their ranking of recommendations doesn't depend on the context of the user's current navigation or on recency effects. We aimed to design a superior (or at least complementary) model-based recommendation algorithm for ResearchIndex that is tuned for the particular user and that takes recent documents into greater account (so as not to lead users too far astray from their current search goal).

Probabilistic models

We assume we're given a data set consisting of time-ordered sequences of document requests. We refer to individual items in the alphabet as *documents*, or simply *items*. For each document, we define its history H as a thus-far-observed (ordered) subsequence of the current sequence. D_{prev} is the last observed document in H. Our task is to predict the next document D_{next} requested by user U given the history H of document requests for U in the present session. We intend to accomplish this task by learning a probabilistic model $P(D_{next} \mid H)$ from the available training data. (To be precise, we should write $P(D_{next} \mid H, Data)$, but for simplicity we omit $Data$ in conditioning.) This formulation enables a number of choices for modeling P.

At the time of our study, users were actively accessing about 15 percent of ResearchIndex's documents, and they accessed more than half of our data set's documents more than once. Thus, we had to fit a distribution on a variable that takes on an order of 100,000 values. We could fit other probabilistic models (for example, the Markov and multinomial mixture models, described later) to the available data directly despite this huge scale, but the maxent model couldn't be feasibly learned over the entire data set due to its high computational cost. So, we developed a greedy hierarchical approach to clustering that provided a feasible implementation. Clustering overcomes sparsity and high data dimensionality by reducing the original problem to a set of smaller subproblems corresponding to clusters. We then recombined the cluster-specific models to make recommendations across the entire data set. Thus, we were able to compare maxent to nonclustering approaches (the mixture models, correlation, and the similarity-based recommenders). This contrasts with our previous work, which only provided within-cluster recommendations.[4]


Searching the World Wide Web (1998) (Make Corrections) (116 citations)
Steve Lawrence, C. Lee Giles
Science

View or download:
nec.com/~lawrence/...chscience98.ps.gz
plg.uwaterloo.ca/~claclark/cs...lg98.ps
Cached: PS.gz PS PDF Image Update Help

From: nec.com/homepages/lawren...pubdl
(more)
(Enter author homepages)

Search engines: irregular indexing, out-of-date, only index a fraction of the web. Metasearch coverage. 320M

Rate this article: 1 2 3 4 5 (best)
Comment on this article

Abstract: The coverage and recency of the major World Wide Web search engines was analyzed, yielding some surprising results. The coverage of any one engine is significantly limited: No single engine indexes more than about one-third of the "indexable Web," the coverage of the six engines investigated varies by an order of magnitude, and combining the results of the six engines yields about 3.5 times as many documents on average as compared with the results from only one engine. Analysis of the overlap... (Update)

Cited by: More
HEADER by the editors FOOTER by the organizers - Evaluating Search Features (2001) (Correct)
Unknown - Crawling The Infinite (Correct)
What Fraction of Images on the Web Contain Text? - Tapas Kanungo Ibm (2001) (Correct)

Active bibliography (related documents): More All
0.0: An Adaptive Web Page Recommendation Service - Balabanovic (1997) (Correct)
0.0: Iterative Type Analysis and Extended Message Splitting.. - Chambers, Ungar (1990) (Correct)
0.0: Assembly Sequences for Polyhedra - Schweikard, Wilson (1995) (Correct)

Similar documents based on text: More All
1.6: Searching the Web: General and Scientific Information Access - Lawrence, Giles (1999) (Correct)
1.0: Efficient Identification of Web Communities - Flake, Lawrence, Giles (2000) (Correct)
0.9: The Power of Play: Efficiency and Forecast Accuracy.. - Pennock, Lawrence.. (2000) (Correct)

Related documents from co-citation: More All
31: The anatomy of a large-scale hypertextual Web search engine - Brin, Page
29: Accessibility of information on the Web (context) - Lawrence, Giles - 1999 - http://www.wwwmetrics.com/
23: Authoritative sources in a hyperlinked environment - Kleinberg - 1997

BibTeX entry: (Update)

S. Lawrence and C.L. Giles. Searching the world wide web. Science, 280(4):98--100 (1998).
http://citeseer.ist.psu.edu/lawrence98searching.html More

```
@article( lawrence98searching,
    author = "Steve Lawrence and C. Lee Giles",
    title = "Searching the (World Wide Web)",
    journal = "Science",
    volume = "280",
```

Figure 1. A typical screen shot of a document details page in ResearchIndex.

Mixture-based models

We implemented and evaluated two mixture-based models to estimate $P(D_{next} \mid H)$: a mixture of Markov models and a mixture of multinomial models.

The mixture-of-multinomials approach essentially prescribes that we disregard sequential information and treat the available document accesses in H as a "bag of items":

$$P(D_{next} \mid H) = \sum_{k}^{N_c} \alpha_k P(D_{next} \mid H, k)$$

$$P(D_{next} \mid H, k) \propto \prod_t \theta_{kt}^{n_t} \qquad (1)$$

where N_c is the number of mixture components, θ_{kt} is the probability that the document number t is accessed, and n_t is the number of times the document number t was accessed in H. We can fit the model using the expectation-maximization (EM) algorithm.[5] The model's number of parameters is linear in the number of documents and the number of mixture components.

In the first-order Markov model, the main assumption is that the current document depends on the history H only through the last observed document in H—that is, D_{prev}. The following equations define the Markov models mixture:

$$P(D_{next} \mid H) = \sum_{k}^{N_c} \alpha_k P(D_{next} \mid H, k)$$

$$P(D_{next} \mid H, k) \propto \theta_{0,k} \prod_t \theta_{(h \to h+1),k} \qquad (2)$$

where the first equation is just a standard equation for the mixture, the second equation uncovers how each component is modeled, $\theta_{0,k}$ is the probability of observing H_0 as a first document in the history, and $\theta_{(h \to h+1),k}$ is the

Table 1. Statistical properties of the experiments' training and test data.

Data properties	Training set	Test set
Total number of users	567,472	212,339
Total number of documents	263,156	233,361
Average requests per user	8.6785	8.0977
Standard deviation of requests per user	31.9832	27.6857
Minimum requests per user	2	2
Maximum requests per user	4,486	3,421

probability of observing a transition from document number h to document number $h+1$ in the history. For $h = |H|$, the document with index $h+1$ is D_{next}. This model can also be learned by using the EM algorithm. The number of parameters is quadratic in the number of documents and linear in the number of components. When $N_c = 1$, the model reduces to a regular Markov model (single-component mixture). Note that the regular Markov model depends only on the so-called bigrams (first-order Markov terms)—that is, the frequencies of pairs of consecutive documents.

The trigger maxent model

We can also model $P(D_{next} | H)$ as a maxent distribution. Our main motivation for using maxent estimation is to combine the attractive features of the Markov and multinomial models. Maxent seems well aligned with other recent efforts to model the recommendation problem as a sequential rather than a static problem. It effectively estimates the probability of the next visited document given the most recently visited document and past indicative documents. To our knowledge, this is the first application of maxent for collaborative filtering and one of the few published formulations that makes accurate recommendations in the context of a dynamic user session.[6]

In particular, we believe that the document D_{prev} requested immediately prior to D_{next} has the most influence on what D_{next} is. Thus, it's essential to model sequence information at least for the pair D_{prev} D_{next}, as the Markov model does. On the other hand, the documents in H beyond D_{prev} also influence D_{next}. We want something more sophisticated than the multinomial (essentially a zeroth-order) model. A higher- (than the first) order Markov model would probably suffice, but we can't afford to build it because of the "curse of dimensionality" (the exponential growth of complexity). Thus, we must restrict ourselves to models that can be reliably estimated from the low-order statistics but still look at the whole H, and this naturally leads us to consider the maxent model.

We select two flavors of low-order statistics or features: *bigrams*, or first-order Markov terms, and triggers. To introduce the long-term dependence of D_{next} on the documents that occurred in the session history, we define a *trigger* as a pair of documents (a, b) such that $P(D_{next} = b \mid a \in H)$ differs substantially from $P(D_{next} = b)$. To measure the triggers' quality and rank them, we compute mutual information between events $E_1 = \{D_{next} = b\}$ and $E_2 = \{a \in H\}$. We then discard 10 percent of low-scoring triggers but retain all bigrams. Note that the quantity and quality of selected triggers depends on H's length. Because the average transaction only has about eight document requests (see Table 1), we choose 10 to be the history's maximum length for finding informative triggers.

The set of features—bigrams and triggers in our case—together with maximum entropy as an objective function can be shown[7] to lead to the following form of the conditional maxent model:

$$P(D_{next}| H) = \frac{1}{Z_\lambda(H)} \exp\left[\sum_{s=1}^{S} \lambda_s F_s(D_{next}, H)\right] \qquad (3)$$

where S is the number of features, F_s are the features, and $Z_\lambda(H)$ is a normalization constant. This ensures that the distribution sums to 1:

$$Z_\lambda(H) = \sum_D \exp\left[\sum_{s=1}^{S} \lambda_s F_s(D, H)\right]. \qquad (4)$$

We must find the set of parameters λ from the following set of equations that restrict the distribution $P(D_{next} \mid H)$ to have the same expected value for each feature as observed in the training data:

$$\sum_H \sum_D P(D \mid H) F_s(D, H) = \sum_H F_s(D(H), H), \quad s = 1, \ldots, S \qquad (5)$$

where $D(H)$ is the document following H in the training data. Equation 5's left-hand side represents the expectation (up to a normalization factor) of the feature $F_s(D, H)$ with respect to the distribution $P(D \mid H)$, and the right-hand side is the frequency (up to the same normalization factor) of this feature in the training data. There exist efficient algorithms (for example, generalized, improved, and sequential conditional iterative scaling algorithms) for finding the parameters λ that are known to converge if the constraints imposed on P are consistent.

Under fairly general assumptions, we can show that the maxent model is a maximum-likelihood model. We use Gaussian smoothing in our experiment because employing a Gaussian prior with a zero mean on parameters λ yields a maximum a posteriori solution that is more accurate than the related maximum-likelihood solution and other smoothing techniques for maxent models.

Reducing dimensionality by clustering

As mentioned earlier, we can fit mixture models to high-dimensional data using the EM algorithm directly. However, depending on the number of selected features, learning the maxent model directly on our raw training data could take months of computation.[8] As we've shown elsewhere,[4] clustering can solve this problem.

Clustering based on user navigation

By clustering based on user navigation patterns, we aim to maximize the probability that once a user requests a document D from cluster c, the user session stays in c (that is, the session will consist only of documents in c). Our intuition is that within a relatively small time frame, most users are interested in a single topic, and if we can find topically related clusters, we should achieve the goal. Here, instead of looking at the contents of individual document or user queries, we take a purely collaborative approach, clustering documents solely on the basis of user navigation sequences.

To come up with such clustering, we scanned the processed training data and, for each pair of consecutively requested documents, computed its count. These statistics are nothing other than the bigram counts. Even though these counts are fairly sparse—that is, there isn't much incoming or outgoing traffic for each document—clustering such data is still challenging.

We tried the PageGather algorithm for clustering the bigrams.[9] The idea behind PageGather is to create an attribute interaction graph based on bigrams and to identify the graph's cliques using a well-known technique such as join-tree clustering. The algorithm returned one large clique and a lot of smaller ones, whereas we'd prefer a set of clusters with more balanced sizes. So, we decided to use a straightforward hierarchical greedy algorithm (see the pseudocode in Figure 2).

The algorithm starts with empty clusters and then cycles through all documents, picking the document pairs that have the current highest joint visitation frequency as prompted by a bigram frequency (lines 1 and 2). If both documents in the selected pair are unassigned, the algorithm allocates a new cluster for them (lines 3 through 5). If one of the documents in the selected pair has been assigned to a previous cluster, the algorithm assigns the second document to the same cluster (lines 6 through 10). The algorithm repeats for a lower frequency n as long as $n \geq 2$.

After the clustering, we can assume that if a user requests a document from the ith cluster $S[i]$, that user is considerably more likely to prefer a next document from $S[i]$ rather than from $S[j]$, $j \neq i$. That is, $P = P(D_{next} \in S[i] \mid D_{prev} \in S[i], Data) \gg 1 - P$. This assumption is reasonable because, by construction, clusters represent densely connected components (in terms of traffic), and the traffic across the clusters is small compared to the traffic within each cluster.

We previously showed that this clustering solves the high-dimensionality problem.[4] We broke individual user sessions into subsessions, each of which consisted of documents belonging to the same cluster. The problem was thus reduced to a series of prediction problems for each cluster.

We also studied the clusters by trying to find out if the documents within each cluster were topically related. We ran code previously developed at NEC Labs that uses information gain to find the top features that distinguish each cluster from the rest. Table 2 shows the top features for some of the created clusters. The top features are quite consistent descriptors, suggesting that a ResearchIndex user is typically interested in searching among topically related documents, at least during one session (as defined earlier). Thus, a straightforward greedy clustering algorithm does find attractive topically related clusters.

```
Input: Bigrams counts $B[i, j]$; Number of Clusters $C$;
Output: Set $S$ of $C$ Clusters.
Algorithm:

     0.  $n_C = 0$;
     1.  set $n = argmax_{i,j} B[i, j]$                 // most frequent bigram
     2.  for all docs $i, j$ such that $B[i, j] == n$ do   // all docs with $n$ transitions
     3.      if ($i$ and $j$ are unassigned and $n_C < C$)
     4.          $S[n_C] \leftarrow \{i, j\}$;
     5.          $n_C$++;                              // new cluster for $i$ and $j$
     6.      else if ($i$ is unassigned)
     7.          $S[n_C] \leftarrow S[n_C] \cup \{i\}$;
     8.      else if ($j$ is unassigned)
     9.          $S[n_C] \leftarrow S[n_C] \cup \{j\}$;
    10.      end if
    11.      $B[i, j] = -1$;
    12.  end for
    13.  if ($n \geq 2$) goto 1
    14.  Return $S$
```

Figure 2. Pseudocode for the hierarchical greedy algorithm.

Combining predictions from different clusters

To make predictions on the raw test data, we need the ability to combine predictions of the learned maxent models for different clusters into a predictor that would work on unclustered data.

We used the following idea borrowed from Joshua Goodman.[8] Consider an arbitrary clustering C of a set of documents such that a document belongs to one and only one cluster. Then, we can represent the probability $P(D_{next} \mid H)$ as

$$\begin{aligned} P(D_{next} \mid H) &= \sum_{C} P(D_{next} \mid C, H) P(C \mid H) \\ &= P(D_{next} \mid C(D_{next}), H) P(C(D_{next}) \mid H) \end{aligned} \tag{6}$$

The first equality is just a rule of full probability; the second equality holds because each document has a unique cluster assignment.

Our scheme for greedy clustering, described earlier, applies directly to this setting. We use a maxent model to represent $P(D_{next} \mid C(D_{next}), H)$ in each cluster. The only remaining question is how to learn the weights $P(C(D_{\text{next}}) \mid H)$. Other researchers suggest learning a separate maxent model to represent the weights.[8] However, we found that the following ad hoc approximation scheme works reasonably well in practice.

Recall that every document has an associated cluster. If we scan H, we can accumulate the frequency with which each cluster (represented by a document in H) has occurred in H. For instance, if all documents in H were from the same cluster c,

Table 2. Top features for some of the clusters in our experiments.

Cluster	Features
1	agent, agents, behavior, good, autonomous, the_agent, an_agent, …
2	training, clustering, distance, classification, kernel, svm, support, …
3	web, documents, query, the_web, queries, pages, engines, …
4	packet, fast, routing, address, the_network, ip, packets, speed, …
5	transform, channel, coding, rate_compression, images, coefficients, …
6	detection, agents, security, intrusion_detection, host, …
7	traffic, rate, packet, long, wide_scheduling, service, qos, …
8	mobile, wireless, protocol, service, services, location, …

only c would get a nonzero weight of 1 and we'd only have to make predictions within c. If half the documents in H were from c_1 and half from c_2, we'd make predictions in each of these clusters and weigh them equally. We used this technique in our experiments.

Similarity-based recommenders

The following similarity recommenders are available in ResearchIndex (more details are available elsewhere[1]):

- *Active Bibliography* finds documents that make similar citations and are also close in the sense of word frequencies.
- *Sentence Similarity* displays documents with a significant number of identical sentences.
- *Text Similarity* recommends documents that resemble the current one in terms of word occurrences.
- *Users Who Viewed* displays other documents requested by the users who viewed the current document.
- *On Same Site* shows other documents listed on the same Web site as the current document.
- *Cited By* lists documents that cite the current one.
- *Co-citation* recommends documents that are often cited together with the current document.
- *Cites* displays documents cited by the current document.

We populated the lists of recommendations for individual similarity-based predictors by taking the information currently stored in the system, which is in most cases three to five top recommendations per document per recommender. The ResearchIndex Merge predictor pulls all similarity-based recommenders together—essentially, this corresponds to the currently available recommending system in ResearchIndex.

The experiments

We compared several probabilistic model-based collaborative recommenders to the standard correlation method and to the similarity-based recommenders ResearchIndex currently uses.

We obtained a log file that recorded more than six months' worth of ResearchIndex data that can roughly be viewed as a series of requests <time, user, document>. We partitioned the log chronologically into approximately five million training requests (covering 159 days) and 1.7 million test requests (covering 50 days). We preprocessed the training and test sets as follows. We assigned each document indexed in ResearchIndex a unique document ID. Whenever a user accessed the site with a cookie-enabled browser, we identified him or her as a new or returning user and recorded all activity on the server side with a unique user ID (UID) and time stamp.

In the first processing step, we aggregated the requests by UID and broke them into sessions. For a fixed UID, we defined a session as a sequence of document requests, with no two consecutive requests more than T seconds apart. In our experiments, we chose $T = 120$, so that if a user was inactive for more than 120 seconds, his next request marked the start of a new session.

We compared several probabilistic model-based collaborative recommenders to the standard correlation method and to the similarity-based recommenders ResearchIndex currently uses.

The next step included heuristics, such as

- Identifying and discarding the sessions belonging to robots (they obviously contaminate human users' browsing patterns)
- Collapsing all same-consecutive-document accesses into a single instance of this document (our objective was to predict what interested the user beyond the currently requested document)
- Getting rid of all documents that occurred just once in the log
- Discarding sessions containing only one document

After preprocessing, we represented the data as a collection of ordered sequences of document requests, possibly with multiple sequences per user (see Table 1 for statistical properties of the data). All models and recommenders took this data as input.

Parameters of competing models

We clustered the training data set using our greedy algorithm. A maxent model was learned separately for each cluster, with all bigrams and 90 percent of the top triggers retained for evaluation. We obtained global predictions of the maxent model on the test data by combining individual cluster maxent models.

We compared our maxent approach with the following models:

- A single-component Markov model
- A mixture of Markov models (30 components)
- A mixture of multinomials (60 components)
- The correlation method—one of the best and most widely used collaborative filtering recommenders
- The individual similarity-based predictors already available in ResearchIndex
- The ResearchIndex Merge similarity-based predictor

For computation reasons, we didn't optimize the models' adjustable parameters (such as the number of components for the mixture or the variance of the prior for maxent models) or the number of clusters (1,000).

Evaluation metrics

We trained all the models on the training data and evaluated them on the test data. We evaluated them by scanning the sequences of document requests document by document and, for each document in the sequence, predicting the next document's identity.

We used the average height of predictions on the test data as the main evaluation criterion. A prediction's *height* is similar to the rank of search results and corresponds to the requested document's rank within the recommendation list of a given model or recommender. We formally define this as follows. Assuming that the probability estimates $P(D_{next} \mid H)$ are available from a model P for a fixed history H and all possible values of D_{next}, we first sort them in the descending order of P and then find the distance in terms of the number of documents to the actual requested D (which we know from the test data) from the top of this sorted list. The height tells us how deep into the list the user must go to see the document of actual interest. The height of a perfect prediction is 0—that is, the requested document is the first one on the list. For ResearchIndex recommenders, we used the order of predictions as stored in the system. If a specific recommender didn't have any predictions for a given document, we set the height to infinity for all occurrences of this document in the test data.

To make comparisons among different recommenders easier, we binned the heights of predictors. We used bins with intervals [0, K] for $K = 1, \ldots, 5, 10$. For all recommenders and all bins, we computed two statistics:

- *Hits.* The total number of requests falling within the bin normalized by the total number of recommendations made
- *Height.* The average height of hits within the bin

The best performing model would place most predictions inside the bins with smaller values of K, and within those bins the average heights would be as low as possible. Note the hits statistic is primary for comparison, while the height statistic plays a role only for ranking recommenders with similar hit ratios.

Our experiments aimed to establish which of the competing recommending strategies provided the best accuracy in terms of hits and height statistics. We also compared the recommenders with respect to the average online time and the total memory taken to generate a prediction. Time was an essential characteristic: the current implementation of ResearchIndex ignores any recommendations that take more than 0.01 second to make. Memory was less important, because in principle an attractively accurate and reasonably fast recommending system can reside on a separate computer, with all its memory resources available for recommending.

We expect that our models won't be competitive with similarity-based recommenders with respect to memory, because the latter use only small lists of predictions per document, whereas the former model attribute interactions and can be quite expensive. For a simple example, consider the case of modeling 250,000 attributes, which is roughly how many we had. All bigrams take quadratic memory in the number of attributes, which would result in roughly $1/2 \times 250{,}000^2 \times 4$ bytes $\approx$ 116 Gbytes. Luckily, as we'll see later, many bigrams have 0 counts and don't need to be stored. On the other hand, if we keep only seven numbers per document (as would be the case for most similarity-based recommenders), this would only result in $250{,}000 \times 7 \times 4$ bytes $\approx$ 6.6 Mbytes.

The time taken to generate recommendations for all D_{next} using a probabilistic model $P(D_{next} \mid H)$ could be quite high. Indeed, this operation amounts to sequentially generating $P(D_{next} \mid H)$ for all D_{next} and sorting the resulting list. An important role in reducing the time requirements for probabilistic models thus belongs to selecting a limited set of candidates for D_{next} depending on H. Because selecting the candidate set can be nonobvious for certain models and straightforward for others, we review our choices here:

- *Markov mixtures.* We can generate the candidate set naturally by assessing the bigrams D_{prev} D_{next} for each document D_{prev}.
- *Multinomial mixtures.* In the absence of other obvious choices, we can choose the candidate set to be the set of all similarity-

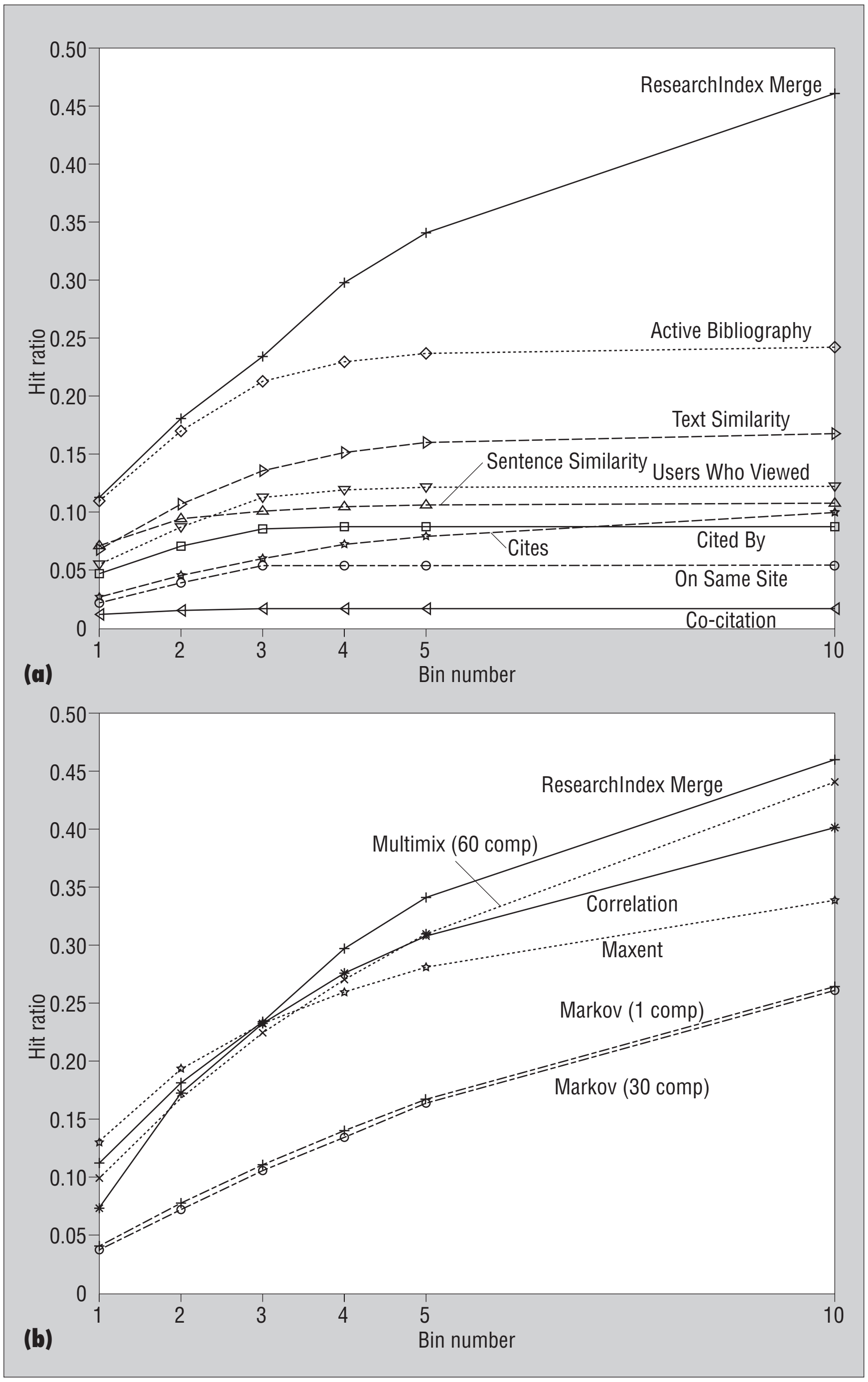


Figure 3. The number of hits for each bin for (a) all recommending methods currently used in ResearchIndex and (b) all probabilistic collaborative-filtering models, correlation, and the ResearchIndex Merge recommender.

based recommendations available for D_{prev}.

- *Correlation.* There is no candidate set to choose (so there are tremendous time requirements as a result).
- *Maxent.* We can define the candidate set naturally by the set of documents belonging to the clusters extracted from *H*.

Thus, we expect our models to be fast enough to fit into the recommending time limit requirements mentioned earlier and accurate enough to compete with existing methods.

Experimental results

The graph in Figure 3a shows the number of hits for each bin for all recommending methods currently used in ResearchIndex. As follows from the plot and in line with our expectations, the ResearchIndex Merge recommender is the best similarity-based recommender. So, we'll use this Merge recommender in the remainder of the discussion to represent ResearchIndex predictors. Figure 3b shows the plot similar to that in Figure 3a but for all probabilistic collaborative filtering models, correlation, and the ResearchIndex Merge recommender.

As Figure 3b shows, the recommender systems behaved differently for small and large values of *K*. Maxent outperformed all recommenders, including ResearchIndex Merge, for $K \leq 3$. It was also one of the top predictors for larger values of *K*, but ResearchIndex Merge, the mixture of multinomials, and correlation models improved more for $K \geq 5$. It's preferable to have better performance when the number of predictions the system can make is small. After all, if the length of the recommendation list were infinite or close to the number of items available, all predictors' hit ratios would be 1.

The mixture of multinomial models appears to be one of the top choices for $K \geq 5$. This isn't surprising because we created the list of candidates for the mixture of multinomials from the predictions of the individual similarity-based recommenders. A single-component Markov model's performance improved for larger bins. Interestingly, the mixture of Markov models failed to improve over the performance of a single component model, probably due to overfitting. The Co-citation was the worst and the Active Bibliography was the best stand-alone similarity-based recommender, whereas Sentence Similarity and Cited By performed better than Co-citation; however, their numbers of hits were still less than half of maxent. Correlation improved its performance for larger values of *K*, scoring one of the best for $K = 10$, but ResearchIndex Merge gave the best hit ratio for all $K \geq 5$.

Table 3 gives the average height of predictions for all recommenders and bins. Note that we treat average height within bins as a secondary statistic, used only for comparing recommenders with similar hit ratios. We prefer lower values of height, unlike with hit ratios. Of the best five competing techniques with respect to hit ratios (maxent, multinomial mixture, correlation, Markov mixture, and ResearchIndex Merge), maxent performed the best with respect to average height, Markov mixture was the worst, and the other three were fairly close. Overall, the model-based approach provided an attractive alternative to current recommenders in terms of the quality of its predictions.

Table 4 compares various models with respect to the average time taken and memory required to make a prediction. The data clearly illustrates that all the models we chose for experiments were on average faster than 1/100 of a second, which is currently the maximum time the system allows for generating a recommendation. In line with our expectations, our models' memory requirements were much higher than that for the individual recommenders. However, as we argued earlier, this shouldn't constitute a practical problem for a recommender with reasonably fast performance running on a stand-alone machine. For the maxent approach, we might further reduce memory consumption by either limiting the maximum size or increasing the number of clusters. In the former case, if we fix the maximum size of the cluster at 1,000, memory usage drops from 458 to 276 Mbytes with a negligible decrease in performance. Increasing the number of clusters will probably also result in less accurate models and will require a better modeling of $P(cluster \mid H)$.

Overall, all the algorithms trade off accuracy, speed, and memory use. Our results demonstrate that the maxent approach offers one of the best performance profiles. It's more accurate than the other model-based approaches, about as accurate as correlation, and orders of magnitude faster than correlation. Maxent even compares favorably against the current similarity-based recommenders in ResearchIndex. This is significant because the test data should be biased in favor of the similarity-based recommenders because they were installed and running when the data was generated.

Most of the models we tested are easy to fit to sparse, high-dimensional data without any additional preprocessing. By clustering documents based on user navigation patterns, we can also apply maxent modeling to the ResearchIndex data. Thus, all the models generated fast enough recommendations to be used in real time on high-volume Web servers. However, the maxent approach gives us a unique advantage—the ability to model long-term interactions and dependencies in data sequences. Maxent did especially well when the recommendation list was limited to a small size. We believe this indicates that maxent can achieve better results in real-time recommendations, since its recommendations would affect users' future requests.

Table 3. Average heights of predictions for bins 1–5 and 10 and four best recommenders with respect to the hits ratio (see Figure 3a).

	Bin					
Model	**1**	**2**	**3**	**4**	**5**	**10**
ResearchIndex Merge	0	0.3785	0.7440	1.2285	2.5814	2.9057
Multinomial mix	0	0.3890	0.7552	1.2125	2.5698	2.9251
Correlation	0	0.5766	0.9441	1.2645	1.5509	2.7208
Maxent	0	**0.3206**	**0.5964**	**0.8448**	**1.0694**	**1.9824**
Markov mix	0	0.4561	0.8962	1.3209	1.7332	3.5504

In any application domain, we can't answer the question "which recommender is the best" solely by doing offline experiments on historical logs as we did in the work reported here. For instance, ResearchIndex enforces a certain interface, showing only the top three documents according to each of the available similarity-based recommenders. This clearly affects user interaction with the system, consequently biasing the logs. In future work, we plan to perform "live" testing of the clustering approach and various models in ResearchIndex. We also plan to enhance our maxent approach with a better model for $P(cluster \mid H)$, which could also be modeled using maxent.[8] Our recent work[10] suggests that for difficult prediction problems, we might improve the plain maxent technique by using mixtures of maxent models. This could yield better accuracy and faster recommendations; however, we still need to explore the feasibility of fitting this model to the data. We also plan to evaluate different clustering methods for documents and try to combine prediction results for different clusterings. We've also started exploring the use of content-based recommendation information for clustering, and clustering based on user queries. Yet another interesting area of research is combining collaborative filtering and content recommendations.

When mining the log file of ResearchIndex, we observed that different groups of users had different patterns of browsing. Some users never followed the recommendations; others used a totally different order of similarity-based predictors. We believe that further personalization of recommendations can boost the system's performance.

Acknowledgments

We thank Steve Lawrence for making available the ResearchIndex log data, Eric Glover for run-

Table 4. Average time per 1,000 predictions and total memory used by various models.

	Time (sec.)	**Memory (Mbytes)**
Cited By	0.0062	3.80
Active Bibliography	0.0082	6.00
Sentence Similarity	0.0056	3.30
Users Who Viewed	0.0078	5.90
Text Similarity	0.0080	6.30
Co-citation	0.0049	2.40
Cites	0.0072	7.20
On Same Site	0.0068	6.00
ResearchIndex Merge	0.0144	24.00
Multinomial mixture, 60 computed	2.5285	60.75
Markov mixture, 1 computed	0.0703	9.73
Markov mixture, 30 computed	0.5204	292.00
Maxent	7.6281	458.10
Correlation	> 1 hour	80.90

The Authors

Dmitry Pavlov is a technical staff member in search at Yahoo! Before that, while working at NEC Research Institute, he created a collaborative-filtering recommending system for CiteSeer and worked on personalization problems. His research interests include data mining and machine learning, particularly the development of scalable algorithms, probabilistic modeling, and applications. He received his PhD in computer science from the University of California, Irvine, and is a coinventor of two patents filed by Microsoft and IBM Research. Contact him at Yahoo Inc., 701 First Ave., Sunnyvale, CA 94089; dpavlov@yahoo-inc.com.

Eren Manavoglu is a PhD student in computer science and engineering at the Pennsylvania State University. Her research interests include data mining, machine learning, and information extraction. She received her MS in computer science and engineering from the Pennsylvania State University. Contact her at 310 Information Sciences and Technology Bldg., University Park, PA 16802; manavogl@cse.psu.edu.

David M. Pennock is a senior research scientist at Yahoo Research Labs. His research interests include information markets, e-commerce, Web analysis and modeling, auctions, recommender systems, machine learning, and artificial intelligence.He received his PhD in computer science from the University of Michigan and has three patents relating to computational issues in e-commerce and the World Wide Web. Contact him at Yahoo! Research Labs, 74 N. Pasadena Ave., 3rd Floor, Pasadena, CA 91103; dp@nnock.com.

C. Lee Giles is the David Reese Professor in the School of Information Sciences and Technology at the Pennsylvania State University. He was one of the creators of the CiteSeer search engine. He received his PhD in optical sciences from the University of Arizona. He is a member of *IEEE Intelligent Systems*' editorial board, a Fellow of the IEEE, and a member of the ACM, American Association for the Advancement of Science, International Neural Network Society, and Optical Society of America. He holds six patents, one winning the NEC Patent of the Year Award, and has applications pending for two more, including patents related to Web search. Contact him at 332 Information Sciences and Technology Bldg., University Park, PA 16802-6823; giles@ist.psu.edu.

ning his naming code on our clusters, and Kostas Tsioutsiouliklis and Darya Chudova for many useful discussions.

References

1. S. Lawrence, C.L. Giles, and K. Bollacker, “Digital Libraries and Autonomous Citation Indexing,” *Computer*, vol. 32, no. 6, 1999, pp. 67–71.
2. B. Sarwar et al., “Analysis of Recommender Algorithms for E-Commerce,” *Proc. 2nd ACM Conf. Electronic Commerce* (EC 00), ACM Press, 2000, pp. 158–167.
3. P. Resnick et al., “GroupLens: An Open Architecture for Collaborative Filtering of Netnews,” *Proc. ACM 1994 Conf. Computer Supported Cooperative Work*, ACM Press, 1994, pp. 175–186.
4. D. Pavlov and D. Pennock, “A Maximum Entropy Approach to Collaborative Filtering in Dynamic, Sparse, High-Dimensional Domains,” *Proc. Neural Information Processing Systems* (NIPS 2002), Bradford Books, 2002.
5. A.P. Dempster, N.M. Laird, and D.B. Rubin, “Maximum Likelihood from Incomplete Data via the EM Algorithm,” *J. Royal Statistical Soc. B*, vol. 39. no. 1, 1977, pp. 1–38.
6. G. Shani, R. Brafman, and D. Heckerman, “An MDP-Based Recommender System,” *Proc. 18th Conf. Uncertainty in Artificial Intelligence*, Morgan Kaufmann, 2002, pp. 453–460.
7. F. Jelinek, *Statistical Methods for Speech Recognition*, MIT Press, 1998.
8. J. Goodman, “Classes for Fast Maximum Entropy Training,” *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing* (ICASSP 01), IEEE CS Press, 2001, pp. 557–560.
9. M. Perkowitz and O. Etzioni, “Adaptive Web Sites: Automatically Synthesizing Web Pages,” *Proc. 15th Nat'l Conf. Artificial Intelligence* (IAAI 98), AAAI Press, 1998, pp. 727–732.
10. D. Pavlov et al., “Mixtures of Conditional Maximum Entropy Models,” *Proc. 20th Int'l Conf. Machine Learning* (ICML 03), AAAI Press, 2003, pp. 584–591.