REFEREE: An open framework for practical testing of recommender systems using ResearchIndex

Dan Cosley
Department of Computer Science and Engineering
University of Minnesota, Minnesota, MN 55455 USA
(612) 624-8372
cosley@cs.umn.edu

Steve Lawrence, David M. Pennock
NEC Research Institute, 4 Independence Way, Princeton, NJ 08540
(609) 951 2676, (609) 951 2715
{lawrence,dpennock}@research.nj.nec.com

Abstract

Automated recommendation (e.g., personalized product recommendation on an ecommerce web site) is an increasingly valuable service associated with many databases—typically online retail catalogs and web logs. Currently, a major obstacle for evaluating recommendation algorithms is the lack of any standard, public, real-world testbed appropriate for the task. In an attempt to fill this gap, we have created REFEREE, a framework for building recommender systems using ResearchIndex—a huge online digital library of computer science research papers—so that anyone in the research community can develop, deploy, and evaluate recommender systems relatively easily and quickly. ResearchIndex is in many ways ideal for evaluating recommender systems, especially so-called *hybrid recommenders* that combine information filtering and collaborative filtering techniques. The documents in the database are associated with a wealth of content information (author, title, abstract, full text) and collaborative information (user behaviors), as well as linkage information via the citation structure. Our framework supports more realistic evaluation metrics that assess user buy-in directly, rather than resorting to offline metrics like prediction accuracy that may have little to do with end user utility. The sheer scale of ResearchIndex (over 500,000 documents with thousands of user accesses per hour) will force algorithm designers to make real-world tradeoffs that consider performance, not just accuracy. We present our own tradeoff decisions in building an example hybrid recommender called *PD-Live*. The algorithm uses content-based similarity information to select a set of documents from which to recommend, and collaborative information to rank the documents. PD-Live performs reasonably well compared to other recommenders in ResearchIndex.

Keywords: ResearchIndex, CiteSeer, collaborative filtering, recommender systems, personalization, content, social filtering, information retrieval, digital libraries

1 Introduction

As the number of databases (e.g., product catalogs) with (limited) web access grows, and the unstructured web "database" itself grows exponentially, finding desired information becomes increasingly difficult. *Recommender systems* tackle such information overload by using the opinions of a group of people to identify information that individuals within the group are likely to find valuable, a methodology which complements more traditional in-

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

formation filtering techniques which analyze the content of documents. Most working applications of recommender systems are in the realm of ecommerce (e.g., recommending products at a web store). For example, Amazon.com alone uses several recommender systems. One gives simple (non-personalized) recommendations: "users who bought *Men Are From Mars, Women Are From Venus* also bought *Space Invaders From Pluto!*". A second system allows a user to post comments about an item, so that other customers can read the comments before they buy. A third system makes personalized recommendations based on the books that a user has bought and any explicit ratings of books (on a scale from one to five) that he or she has entered.

The third Amazon recommender described above is an example of a pure *collaborative filtering* (CF) system that computes recommendations based only on the ratings (explicit or implicit) given by users, ignoring product attributes. As a result, CF systems are completely domain independent. However, CF systems can fail when data is too sparse, when recommending new items, or when recommending to new users. More sophisticated recommender systems combine user ratings with domain-specific descriptive information about the items—after all, there is no reason to throw the bag-of-words out with the trash. Using content information is a promising way to overcome sparsity and the new-item and new-user problems (sometimes referred to as the *cold-start* problem [23]).

A serious impediment to progress in recommender systems research is the lack of a standard framework for evaluating competing algorithms in a real-world setting. We argue that ResearchIndex can serve as an excellent testbed for recommendation algorithms, especially hybrid algorithms that combine collaborative and content information. We have developed the **RE**commender **F**ramework and **E**valuator for **RE**s**E**archIndex, or REFEREE. REFEREE allows anyone to quickly implement, field, and evaluate recommenders for documents in ResearchIndex. Competing systems can be evaluated according to actual user behavior (click-through rates and download rates), rather than measures like predictive accuracy that may or may not translate into user satisfaction. We believe that REFEREE has the potential to stimulate research in at least three areas: (1) recommender systems that merge content and collaborative data, (2) systems that must make tradeoffs to sacrifice accuracy and elegance for speed and memory savings, and (3) systems that recommend *in context*, meaning that recommendations depend on current user actions. We hope that REFEREE will also stimulate research in areas that we have not contemplated. To start the ball rolling, and to provide a skeleton system for other researchers, we have developed our own hybrid recommender, called *PD-Live*. PD-Live uses a novel method for combining content and collaborative information that does take context into account, allowing it to make most recommendations from among the over 500,000 documents in ResearchIndex within 200 milliseconds. We present preliminary results showing how it compares with baseline systems and the content-based recommenders already provided with ResearchIndex.

2 Related work

2.1 A brief history of CF

The term collaborative filtering was introduced by Tapestry [8], although they used it in the broader sense usually denoted by "recommender systems" today. Tapestry saw documents as structured entities (their model was email) and users could create structured queries, not unlike today's email filters. For example, a user could say to ignore any mail with a subject containing "toner", or from a sender whose address ended in "hotmail.com". In addition, users could add annotations—votes, ratings, text, etc.—to a document and create queries to operate on others' annotations. This approach gives users great control and flexibility in filtering information, but it has two drawbacks. First, the user must manually create the filters. Second, Tapestry is best suited for small groups where people know each other. Tapestry queries were often of the form "documents that Mark likes"; this requires that you know Mark, or more generally, that you know the people who are like you and whose opinions you should value. It also requires enough community cohesion so that, for example, Mark doesn't mind if you know what he likes.

Automated collaborative filtering addresses these drawbacks by using a model of recommendations in which *users* assign *ratings* to *items*. The system uses these ratings to find other users who are most similar to a given user, and uses their opinions to make recommendations. GroupLens [21] uses this approach to filter Usenet news, while other systems have used this approach to recommend items from music [26] and movies [10] to web pages [1] and jokes [9].

2.2 CF algorithms

The original algorithms used similarity metrics computed between two users' ratings of items, where users explicitly entered their ratings as values on a Likert scale. In GroupLens, for example, users were asked to rate Usenet news articles on a scale from 1 (very bad) to 5 (very good). Common similarity metrics used include Pearson correlation [21], mean squared difference [24], and vector similarity [5]. The system chooses a set of neighbors most similar to the user it is computing predictions for; typically, the system will choose as neighbors the n most highly correlated users, or all users with a similarity score over some threshold. It then computes predicted ratings on items the user has not yet seen based on his neighbors' ratings for those items. The system sorts the items by predicted rating and presents them to the user in that order. Pearson correlation works reasonably well and is quick to compute, making it the dominant algorithm used in deployed systems today.

Researchers in machine learning have suggested that the recommendation problem can be cast as the classic problem of classification. They have applied a number of machine learning techniques, including inductive learning [2], clustering [26], neural networks [3], and Bayesian networks [5]. Recent approaches tend to employ probabilistic models. Personality diagnosis [18] assumes that users report ratings with error and forms neighborhoods by computing the probability that a given user is of the same type as other users in the system. Probabilistic clustering [14] is similar, except that it assumes that users fall into a fixed number of types and uses Expectation Maximization (EM) to estimate the probabilities of a user being of each personality type based on their ratings.

In general, CF algorithms based on machine learning techniques perform as well as or slightly better than the original correlation methods. Many of these methods have the advantage that they compute a user model and so make very fast predictions online, although the cost for building the models must be paid at some point. Pennock et al. [17] explore axiomatic justifications and theoretical limitations which apply to CF algorithms.

2.3 Hybrid systems: combining content with CF

One of the original motivations for collaborative filtering was to complement traditional approaches from the field of information retrieval. These approaches use the content of the items to make filtering decisions, which is sometimes difficult. In some domains, such as movies and music, there is little machine-understandable content. In other cases, such as literature, personal taste dominates the notion of relevance. In any case, making recommendations by using ratings independent of content is a general technique that can be used in any domain [11].

However, it seems wasteful to throw away content information when it is available. Using content-based information also allows recommender systems to overcome startup problems when the available user-item interaction data is sparse or, as an extreme case, when a user or an item is new to the system. Researchers have tried several different hybrid systems which combine content-based and collaborative information.

The *filterbot* model developed by Sarwar et al. [22] and Good et al. [10] injects content-based information into a collaborative filtering environment. Filterbots are agents that act as users of a recommender system which rate all items that enter the system. These agents use rules based on the content, such as the amount of quoted text for Usenet posts and the genre for movies, to determine their ratings, allowing them to introduce machine-generated content-based similarity judgments into a recommender system.

Another class of systems use the content-based information and collaborative information separately. Claypool et al. [6] developed P-Tango, an online newspaper which combines the results of two separate recommenders, one content-based and one which uses CF. The system merges the results from the two recommenders, assigning more weight to the one which performed better for a given user. ResearchIndex itself uses several separate recommenders, some of which use content-based and one of which uses collaborative information [4]. Instead of combining the results of the recommenders, however, it simply presents each recommender's results separately and allows the user to decide which recommenders are working well.

Other approaches combine the notions of content-based and collaborative similarity. Several systems use preferences to build keyword-based user models and then recommend using the models, such as Fab [1]. Basu et al. [2] found that they could derive artificial features that merge content-based and collaborative information, then train the Ripper learning system [7] to outperform a collaborative recommender. Pazzani uses data from the Syskill and Webert system to explore using content-based, collaborative, and demographic information to make recommendations [16]. Popescul et al. [19] extend Hofmann's probabilistic model [12] to create a model that relates words, documents, and users' preferences for documents through a set of implicit topics.

3 A standard testbed for evaluating recommender systems in situ

Several standard datasets, such as the EachMovie dataset of movie ratings [15], exist for testing pure CF algorithms. This is not the case for hybrid recommenders. The papers described above which explore hybrid systems use different datasets and metrics, making them hard to compare. Standard machine learning datasets are not useful as they have no collaborative information, so researchers have typically taken a CF dataset and grafted content onto it from the outside. In the case of movies, for example, researchers might supplement EachMovie with information from the Internet Movie Database (http://www.imdb.com/). Even when compared against a common dataset, the best offline algorithms may not be the best recommender systems in real-world settings. This is because the algorithm may be too computationally inefficient, and/or the offline evaluation criteria (usually predictive accuracy) may not reflect true user sentiment.

In this section we describe REFEREE, our framework for implementing recommender systems in the context of ResearchIndex. We describe the resources it provides to developers. We also argue for metrics that, unlike typical accuracy metrics, evaluate recommenders based on their impact on user behavior.

3.1 What is ResearchIndex? Why use it?

ResearchIndex, also known as CiteSeer, is an autonomous citation indexing system [13] which indexes a large fraction of the computer science literature available on the web. The system locates papers, downloads them, and automatically extracts information such as abstracts, authors, and citations. It also computes several forms of similarity between documents, including citation links both to and from a document, sentence similarity, and an Amazon-like "users who viewed x also viewed y" similarity [4]. Users can perform keyword searches on citations or documents, and can navigate between documents using the similarity measures computed by the system. Figure 1 shows a typical document details page in ResearchIndex. Document details pages show users meta-information about a document, such as its title, authors, and abstract, along with ways to download the document. These pages also display a number of links to other documents, generated by recommenders that use the similarity metrics mentioned above.

We believe that ResearchIndex is an excellent testbed for investigating hybrid recommenders for a number of reasons. The domain of research papers has both objective ("this work is related to mine") and taste ("I think this work is interesting") components, suggesting that using both content-based and collaborative similarity will be valuable. ResearchIndex makes it easy to observe a user's behavior, automatically assigning user IDs (as well as allowing users to create user IDs they can use across computers). Recommender system developers can install a local copy of ResearchIndex and mirror the actual site, using a rich API to access enormous amounts of content information about documents in the database. ResearchIndex also has a large, active system, which will allow recommender systems developers to ensure that their algorithms scale well to systems which demand quick response time while servicing thousands of requests per hour, over a database of hundreds of thousands of users and items. Finally, ResearchIndex will provide a standard, well-defined recommendation task that researchers can use to evaluate their algorithms and systems.

3.2 ResearchIndex terminology

The basic entities in ResearchIndex are users, documents, citations, and citation groups. Users are the people and robots who make queries and access documents and citations. ResearchIndex generates user IDs (UIDs) and stores them in cookies. As mentioned above, users who want to carry an identity between machines can create explicit accounts. Documents are the actual research papers that are downloaded and processed. Each document is assigned a document ID (DID). Documents that are found in multiple locations may wind up with multiple document IDs, although ResearchIndex attempts to unify them. Citations are the actual text of citations extracted from the documents. Each citation receives a unique citation ID (CID). Citation groups relate individual citations that cite the same document, with each citation group receiving a group ID (GID). Since ResearchIndex does not store all cited documents (indeed, many cited documents are not available on the web), GIDs are distinct from DIDs.

ResearchIndex calls user actions "events". Most events happen when a user interacts with a document or citation, or issues a query. There are about 30 kinds of events; some of the events most likely to interest recommender systems developers are shown in Table 1.

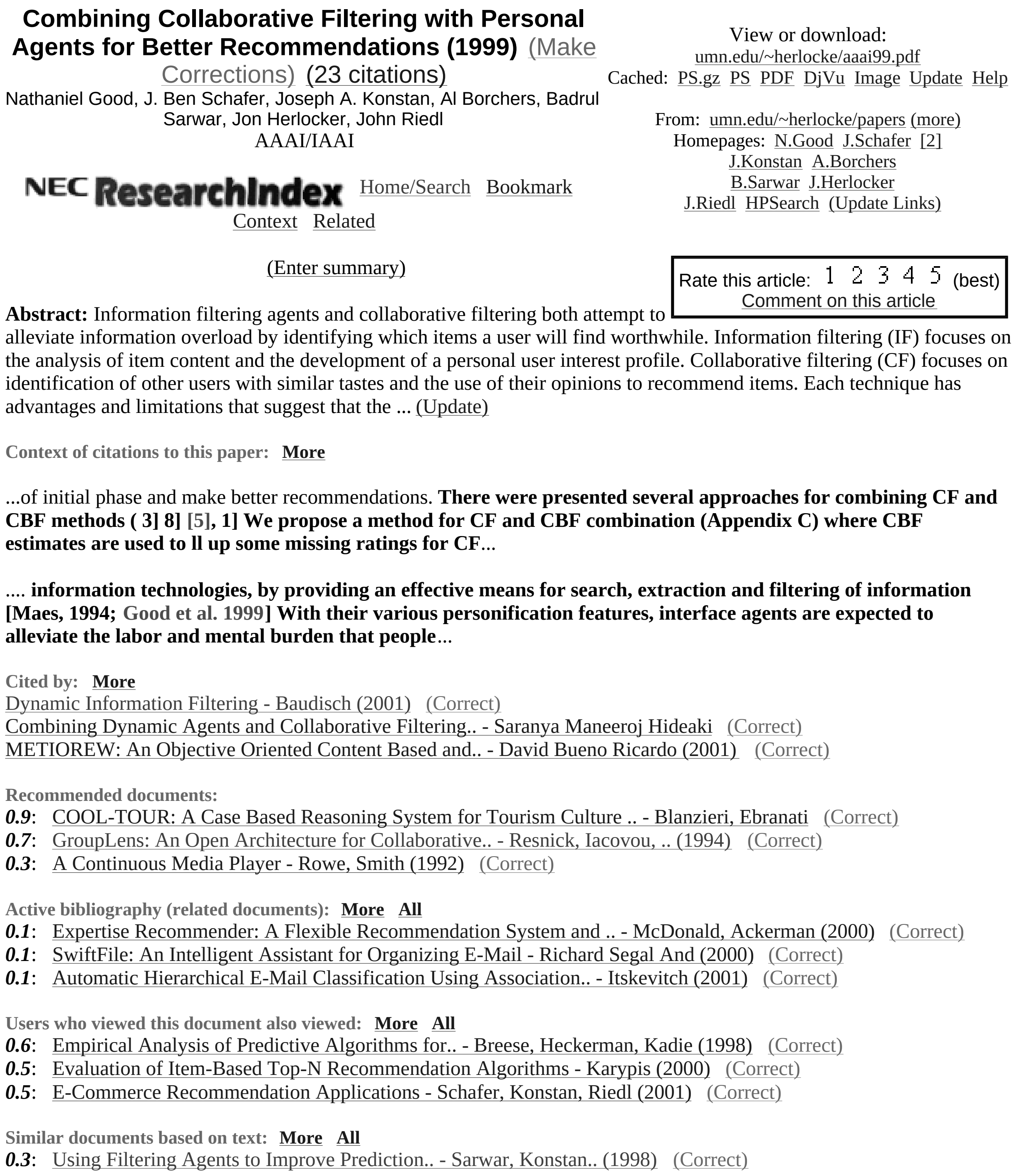
Note that REFEREE has no built-in notion of how important or interesting an event is. Instead, recommender systems developers have to infer what an event signifies about a user's preferences. This is also known as using *implicit ratings*, as opposed to *explicit ratings*, where the user effectively says "I like item x *this* much". Implicit ratings play an important role in ecommerce recommenders, as it is a burden on users to ask for explicit ratings. ResearchIndex does provide a way for users to give a rating on a 1 to 5 scale; however, users give very few explicit ratings compared to the number of events which users generate.

3.3 REFEREE

REFEREE's main purpose is to help recommender systems communicate with ResearchIndex. A Recommender Framework Daemon (RFD) runs on the ResearchIndex site and allows recommenders to receive user events. It also requests recommendations of the form "user x is looking at document y: what three documents might x want to look at next?" The system generally issues between one and five recommendation requests per second, and usually requires recommenders to respond within 200 milliseconds. Figure 2 shows the REFEREE architecture.

In addition to providing a standard testbed, a major goal in the construction of REFEREE was to minimize the amount of work recommender system developers must do in order to implement and test working systems. Skeleton clients in C and Perl manage connections to ResearchIndex and call hook methods when they receive user events and recommendation requests. Developers can implement their system within these methods, or extract the event/recommendation data and forward it to their already-working system. The skeleton client also handles catching up on user events which occur while a recommender is offline.

In addition to the skeleton client, developers can also start from a complete recommender written in C++ called *PD-Live*. PD-Live does all the translation from



Combining Collaborative Filtering with Personal Agents for Better Recommendations (1999) (Make Corrections) (23 citations)
Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, John Riedl
AAAI/IAAI

NEC ResearchIndex Home/Search Bookmark Context Related

View or download:
umn.edu/~herlocke/aaai99.pdf
Cached: PS.gz PS PDF DjVu Image Update Help

From: umn.edu/~herlocke/papers (more)
Homepages: N.Good J.Schafer [2] J.Konstan A.Borchers B.Sarwar J.Herlocker J.Riedl HPSearch (Update Links)

(Enter summary)

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

Abstract: Information filtering agents and collaborative filtering both attempt to alleviate information overload by identifying which items a user will find worthwhile. Information filtering (IF) focuses on the analysis of item content and the development of a personal user interest profile. Collaborative filtering (CF) focuses on identification of other users with similar tastes and the use of their opinions to recommend items. Each technique has advantages and limitations that suggest that the ... (Update)

Context of citations to this paper: **More**

...of initial phase and make better recommendations. **There were presented several approaches for combining CF and CBF methods (3] 8] [5], 1] We propose a method for CF and CBF combination (Appendix C) where CBF estimates are used to ll up some missing ratings for CF**...

.... **information technologies, by providing an effective means for search, extraction and filtering of information [Maes, 1994; Good et al. 1999] With their various personification features, interface agents are expected to alleviate the labor and mental burden that people**...

Cited by: **More**
Dynamic Information Filtering - Baudisch (2001) (Correct)
Combining Dynamic Agents and Collaborative Filtering.. - Saranya Maneeroj Hideaki (Correct)
METIOREW: An Objective Oriented Content Based and.. - David Bueno Ricardo (2001) (Correct)

Recommended documents:
0.9: COOL-TOUR: A Case Based Reasoning System for Tourism Culture .. - Blanzieri, Ebranati (Correct)
0.7: GroupLens: An Open Architecture for Collaborative.. - Resnick, Iacovou, .. (1994) (Correct)
0.3: A Continuous Media Player - Rowe, Smith (1992) (Correct)

Active bibliography (related documents): **More** **All**
0.1: Expertise Recommender: A Flexible Recommendation System and .. - McDonald, Ackerman (2000) (Correct)
0.1: SwiftFile: An Intelligent Assistant for Organizing E-Mail - Richard Segal And (2000) (Correct)
0.1: Automatic Hierarchical E-Mail Classification Using Association.. - Itskevitch (2001) (Correct)

Users who viewed this document also viewed: **More** **All**
0.6: Empirical Analysis of Predictive Algorithms for.. - Breese, Heckerman, Kadie (1998) (Correct)
0.5: Evaluation of Item-Based Top-N Recommendation Algorithms - Karypis (2000) (Correct)
0.5: E-Commerce Recommendation Applications - Schafer, Konstan, Riedl (2001) (Correct)

Similar documents based on text: **More** **All**
0.3: Using Filtering Agents to Improve Prediction.. - Sarwar, Konstan.. (1998) (Correct)

Figure 1: A typical document details page in ResearchIndex includes meta-information, ways to download the document, and recommendations for other documents to pursue. These recommendations are generated by a number of recommenders built into ResearchIndex, as well as by recommenders using the REFEREE framework.

Event name	Happens when	Parameters
documentdetails	User visits a document's details page	UID, DID, related doc info
download	User downloads a document	UID, DID
rate	User assigns an explicit rating	UID, DID, rating
cachedpage	User is viewing document pages	UID, DID, pagenumber
documentquery	User searches documents for a query string	UID, query

Table 1: A sample of user actions in ResearchIndex. Most actions involve a user interacting with a document or citation in the database.

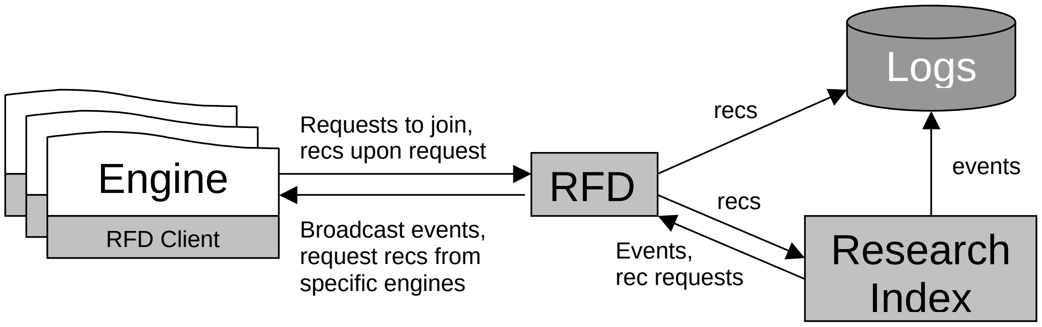


Figure 2: An overview of the REFEREE architecture. Recommender engine writers need only write the Engine portion, making use of the shaded components which REFEREE provides.

ResearchIndex events into users, items, and ratings; maintains an in-memory sparse-matrix representation of the ratings matrix; remembers its state between runs; and can compute most recommendations in under 200 milliseconds using a variant of the personality diagnosis (PD) algorithm of Pennock et al. [18].

In order to develop a REFEREE-based recommender, a researcher would contact ResearchIndex for access to the framework, codebase, and optionally source data. The researcher would then develop a recommender, testing it locally against an instance of ResearchIndex that the researcher would run. Once the recommender is ready, the researcher would change the client to point at the actual ResearchIndex site, and it would then go live. The recommender can run on any machine, as long as the network latency plus computation time required to make recommendations is within the deadline which ResearchIndex imposes. Researchers can make arrangements to run their recommenders on a ResearchIndex machine if necessary.

3.4 Metrics which measure user behavior

Unlike most prior research on recommender systems, REFEREE does not use metrics which focus on retrospective prediction accuracy on a static dataset. Instead, it uses metrics that rely on how users actually behave when viewing recommendations. The metrics borrow from the ecommerce notions of click-through and purchases. These metrics match how users experience ResearchIndex: users "click through" when they follow a recommended link to a document's details page, and "purchase" the document when they download it. We also define "click soon" and "purchase soon" metrics, where users view details for or download a recommended document within a short number of actions after the recommendation. These metrics give recommenders credit for suggesting good documents, even though the user found them through another path or came back to them after a short diversion.

4 Accuracy is not the whole story

We prefer metrics which measure how users respond to recommendations over prediction accuracy metrics such as Mean Absolute Error (MAE). Absolute error is the absolute value of the difference between a user's rating for an item and the system's prediction of how much the user would like the item; MAE is the average error over a set of ratings. Accuracy metrics have their place in offline, retrospective analysis of static datasets, as they may be the best one can do in that situation.

However, accuracy does not tell the whole story. The actual accuracy achieved by an algorithm is poorly defined, since user ratings are neither precise nor consistent. The quest for accuracy may also be misguided, since users do not need precise predictions in order to help guide their decision making. Analysis of static datasets ignores the fact that recommender systems are decision support tools, not the classification systems that many researchers model them as (e.g., [2]). Finally, recommenders affect users' behavior when they make suggestions, an effect that is difficult or impossible to model with static datasets. We discuss these objections to accuracy-based metrics in detail below.

User ratings are not precise—they depend on a person's mood, the time of the recommendation, what other items the user has seen recently, etc. As an example, Hill et al. [11] asked users to re-rate a set of movies they had rated six weeks earlier, and found that the Pearson r correlation between the ratings was 0.83. Statistically speaking, this is a fairly strong correlation; however, the inconsistencies are enough to cloud whether the small accuracy improvements often reported in the literature are in fact meaningful. User ratings are also often not explicit; the system must instead infer a rating from user behavior. These features of user ratings lend a quixotic cast to the quest for prediction accuracy. It is also an open question whether users would even notice a difference between two systems, one of which had an MAE of 0.75, and one which had an MAE of 0.72. Turpin and Hersh [25], for instance, found that users performed question-answering tasks with a baseline text search engine just as well as they did with a much better search engine.

In most cases, absolute prediction accuracy is not needed. Recommender systems are decision support tools: they help users make a decision whether or not to pursue an item. It is not so important that a system computes the exact score a user might give to an item (although a confidence score that indicates how strongly the system believes in its recommendation would be useful). It is also not important for the system to make recommendations that give the user little additional information. For example, a system might recommend a famous research paper that everyone knows about already. Such a recommendation tells the user little. Instead, it is important that the items a system recommends help a user make a decision about what item to choose. A reasonable way to measure this is to see what impact the recommender has on users' behavior. MAE does not measure this impact.

Finally, a recommender system actually changes users' behavior. By suggesting a set of items, a recommender provides a user with a set of choices which the user might not otherwise have made. Breese et al. [5] note that users are more likely to rate items that they like or which the system presents. This means that recommender systems influence which items users consume (and thus can rate) and might even influence the actual ratings users give ("the system recommended it, so it has to be good"). Neither of these important cases can be captured by evaluating recommenders using accuracy metrics on a static dataset.

Despite the above critique, static datasets are a valuable tool for evaluating recommendation algorithms. Static datasets make for convenient experiments. Using these datasets to weed out unsuitable algorithms can also increase the effectiveness of online experiments, as suggested by Rashid et al. [20]. Finally, by providing a fixed and unchanging world, static datasets allow researchers to compare the prediction accuracy of different algorithms more easily than they could with a live data stream. These are all virtues of static datasets and accuracy metrics; we simply believe that retrospective analyses of accuracy fail to tell the whole story about recommender system algorithms.

In any case, ResearchIndex captures raw data: it logs user events, recommendation requests, and documents recommended. We will create anonymized, static datasets which can be used with prediction accuracy metrics as well.

5 PD-Live: An example recommender

To show that REFEREE is a usable framework, we built an example hybrid recommender, PD-Live, which we make freely available to developers. As mentioned above it keeps a sparse-memory matrix of users, items, and ratings, and uses a variant of the personality diagnosis (PD) algorithm to make predictions. We chose PD because we had a reasonably fast working offline version, and the algorithm has been shown to be more accurate than standard correlation methods on EachMovie and ResearchIndex data [18]. It took one programmer about three days to implement the algorithm using the framework, and another two weeks to achieve performance good enough to meet the speed requirements which ResearchIndex imposes. Below we describe our strategy for exploiting both content and collaborative similarity, as well as some compromises required to make the recommender fast enough to deploy.

5.1 Recommending in context: a strategy for building hybrid recommenders

Consider a customer at an electronics store's ecommerce site who has requested information on a portable CD player. The site would like to place a personalized recommendation on the page containing the information. The site's CF recommender finds the customer's neighbors, looks through all the items in the database and suggests…a bagel toaster. This is probably not a good recommendation at this time, even if it is the item the system believes that the customer would rate highest.

Amazon.com (and ResearchIndex) already have non-personalized recommenders to address this problem, in their "users who bought x also bought…" lists. However, we would like to use our knowledge of the user to personalize these recommendations. We call this recommending *in context*: given that the user is viewing information about *Goto Considered Harmful*, what papers should we recommend? The recommender must balance its overall knowledge of the user with some notion of what the user is interested in right now. For example, the original GroupLens recommender treated each Usenet group as a separate set of items [21], so that users viewing recipes would not be recommended jokes or Microsoft flames.

In general, however, most CF recommenders do not consider context explicitly, instead drawing recommendations from all the items they know about. They get some implicit context from the fact that people tend to like similar types of items (e.g., if Dan likes mostly science fiction movies, a recommender will tend to match him up with other sci-fi fans and recommend science fiction rather than romances). They also attempt to recommend serendipitous items—items which are not within the usual scope of a user's interests. This approach is problematic for a recommender for ResearchIndex. First, computing a recommendation from among 500,000 items presents a problem of scale. Second, researchers typically have several areas of interest, meaning that documents with the highest predicted scores will often be in fields that the user is not currently interested in.

Our approach is to use the content-based similarity information that ResearchIndex gleans when extracting citations from documents in order to restrict the items that the recommender considers. Citation links and other similarity measures form a directed graph with documents as the nodes and similarity relationships as the edges. PD-Live does a breadth-first search from the document a user is currently looking at to select a candidate set of documents. It then uses the PD algorithm to generate predictions for these documents and conducts a lottery biased toward higher-scoring documents to select which ones to recommend. If the user is new, or no neighbors can be found who have rated a document, the system falls back to non-personalized average ratings for the documents. If the document is new and no similarity information is available, the system chooses the set of recommendable documents randomly.

This two-phase approach has several nice properties. First, ResearchIndex has already generated the content similarity information, so it is easy and fast to access and use. By using the content-based information separately from the collaborative, we don't have to find a way to combine or weigh these disparate notions of similarity. Using content to select a set of plausible documents and then collaborative information to rank them is also a reasonable model of how people often make decisions: eliminate obviously wrong alternatives and then choose the best among them. Finally, this approach fits well with the notion of relevance adopted by the IR community. In the TREC information retrieval competitions, relevance is assumed to be binary: either a document is relevant or not relevant to a topic. This allows for comparisons between different systems, but simplifies the notion of relevance, which has degrees, varies from person to person, and varies with a person's situation [27]. Our recommender starts with the impersonal, binary notion of relevance, then uses CF to include personal notions of relevance.

5.2 Compromises

In order to produce recommendations in a reasonable amount of time, PD-Live makes a few compromises. As discussed above, it only follows the similarity graph for a short distance, recommending from a small subset (typically around 500) of the 500,000 documents known to ResearchIndex. It also reduces the number of documents considered by only recommending documents that have at least five ratings, since documents rated by only a few users would seldom be recommended anyway.

It also eats memory voraciously, trading space for speed. This is most obvious in the case of ratings; it currently maintains all ratings in memory. Locating potential neighbors (those users who have rated at least one item in common) is time-consuming, so the system caches this information when a user begins a session and only updates it every 15 minutes. For users with a large number of potential neighbors, the cache considers potential neighbors over specific document ranges (e.g., neighbors for documents with DID 0-50000, 50001-100000, etc.). As the number of potential neighbors rises, the document range narrows. This allows the prediction algorithm to consider fewer potential neighbors for each prediction it makes, again gaining speed at the cost of memory.

PD-Live also makes use of only some user actions: those where the user is interacting with a document already stored in the system. Users also take actions when exploring ResearchIndex which pertain to documents not in the database, such as viewing citations made by documents in the database. It would be possible (and probably desirable) to recommend the documents these citations point to as well, although users would have to go to other sources to download the actual documents.

6 Baseline results

We fielded several recommenders in the course of testing REFEREE, including a random recommender (SimRandom) and two versions of PD-Live. We present preliminary measurements of the quality of the recommendations produced by these recommenders, and compare their results to those of the similarity-based recommenders built into ResearchIndex.

6.1 Recommenders

ResearchIndex has a number of built-in recommenders, most of which are content-based.

- **Sentence Overlap**: recommends documents with significant sections of identical text to the current document.

- **Cited By**: documents which cite the current document.

- **Active Bibliography**: documents that make similar citations (the algorithm is similar to bibliographic coupling).
- **Users Who Viewed**: documents that the userbase as a whole have seen, if they have seen the current document.
- **Co-citation**: documents that are often cited together with the current document.
- **On Same Site**: documents found on the same web page as the current document.

We also fielded three recommenders that used REFEREE.

- **PD-Lottery**: PD-Live as described earlier.
- **PD-Top**: PD-Live, except that instead of holding a weighted lottery, it always recommends the highest-scoring documents.
- **Sim-Random**: Selects documents to recommend using the similarity graph, as PD-Live does, but assigns random scores to these documents.

Users receive recommendations in ResearchIndex when looking at a document details page. Each of ResearchIndex's recommenders may suggest up to three documents, and the order of recommenders is fixed—they always appear in the same relative order on the page. This is because a user interface which permutes the order of the recommenders could be difficult for users. There is only one slot on the page for REFEREE-based recommenders, so when multiple such recommenders are running, the framework issues requests for recommendations to each system in round-robin order.

Sometimes two recommenders will suggest the same document. In this case, the recommendation is displayed only once, by the first recommender to appear on the page. This gives an advantage to recommenders which appear earlier on the page (as does the fixed order of presentation). This limits our ability to make comparisons between REFEREE-based recommenders and the built-in ResearchIndex recommenders. However, since the REFEREE-based recommenders will always appear in the same position on the page, and since each page shows recommendations from only one such recommender, comparisons between these recommenders are reasonable.

6.2 Metrics

Our metrics measure whether the user accepted a recommendation. We distinguish between a user looking at a document's details page, and a user actually downloading the document. *Click-thru* measures how often a user immediately follows a link to a document's details page suggested by a given recommender. *Buy-now* measures how often the user downloads a recommended document immediately after clicking through to that document. *Click-soon* and *buy-soon* are similar to click-thru and buy-now, except that they give credit to a recommender for documents it recommended and that the user accessed within the next ten actions (e.g., the user came back to a recommendation after following a different one).

6.3 Results

Table 2 shows the performance of the prototype recommenders we developed along with the performance of ResearchIndex's similarity-based recommenders. These data represent about one day's worth of user activity in ResearchIndex.

For each metric, we present both the raw number of recommendations made by the recommender and followed by the user, as well as the percentage of total recommendations which the user followed. For the most part, the four metrics produce the same rank ordering of the recommenders. The best PD-Live-based recommender places in the middle of the pack, outperforming three of ResearchIndex's built-in recommenders (On Same Site, Co-citation, and Users Who Viewed) while lagging behind three others (Sentence Overlap, Cited By, and Active Bibliography). Note that the percentages for click-thru do not add up to 100% because there are many other actions that users can take besides following a recommendation link.

6.4 Discussion

The Sentence Overlap and Cited By recommenders do much better than the rest. We believe this happens for two reasons. First, these recommenders often produce documents related to the current document and that are more recent. Sentence Overlap allows researchers to find the most recent (and often, most detailed) paper describing a particular piece of research. Cited By locates papers that build on (or at least, come after and criticize) the current paper; this helps researchers to find newer papers related to research of interest. Second, these are the first two recommenders displayed on a document details page. This shows that an important next step for the framework is to come up with a way to account for the placement order of recommendations.

Within the recommenders we built, the PD-Live based recommenders outperform Sim-Random. This shows that personalization adds something to a random selection of nearby documents in the similarity graph, which in turn lends support to our belief that hybrid systems can perform well. PD-Lottery outperforms PD-Top. We believe that this occurs because documents that receive high scores often are based on similarity with only one or two other users—in other words, these are low-confidence recommendations. It is also possible that

Name	Recs	Click-thru	Click-soon	Buy-now	Buy-soon
Sentence Overlap	37637	1741 (4.63%)	3080 (8.18%)	277 (0.74%)	808 (2.15%)
Cited By	83816	1896 (2.26%)	4256 (5.08%)	385 (0.46%)	1212 (1.45%)
Active Bibliography	162003	2316 (1.43%)	5076 (3.13%)	481 (0.30%)	1544 (0.95%)
PD-Lottery	36559	338 (0.92%)	773 (2.11%)	60 (0.16%)	247 (0.68%)
Users Who Viewed	155741	1377 (0.88%)	3190 (2.05%)	293 (0.19%)	985 (0.63%)
PD-Top	37114	324 (0.87%)	742 (2.00%)	53 (0.14%)	229 (0.62%)
Co-citation	30433	225 (0.74%)	565 (1.86%)	59 (0.19%)	190 (0.62%)
On Same Site	140170	875 (0.62%)	1897 (1.35%)	157 (0.11%)	543 (0.39%)
Sim-Random	34027	202 (0.59%)	390 (1.15%)	21 (0.06%)	111 (0.33%)

Table 2: Performance of hybrid and content-based recommenders in ResearchIndex. Percentages represent how often a user followed a link or downloaded a document recommended by the given recommender.

users who visit the same document several times may receive different recommendations with a lottery, giving PD-Lottery the opportunity to recommend more documents.

Apart from the dominating performance of Sentence Overlap and Cited By, PD-Live performed reasonably well. We see several ways to improve upon PD-Live. One easy fix would be to detect robots and exclude them as neighbors. Google's web crawler turns out to be a good neighbor for many users, which is probably not conducive to accuracy.

Another improvement would be to abandon poor similarity measures when building the graph. PD-Live uses similarity information provided by the poorly-performing Co-citation and On Same Site recommenders. These measures have serendipity going for them: a collection of papers on the same site might draw from several research areas, and users may discover interesting research areas they have not considered. However, most searching on ResearchIndex probably involves finding papers relevant to a specific, current project with a conference deadline of tomorrow. Serendipitous discovery is often considered an advantage of collaborative filtering systems over content-based systems; when recommending in context, perhaps it is less of an advantage.

Finally, PD-Live currently only scratches the surface of the content available. In principle, we could use IR techniques on document text to compute similarity or cluster documents. This information could be included in the similarity graph and lead to consideration of relevant documents that ResearchIndex's built-in recommenders do not discover. It is possible that much of the interesting document similarity information is encoded in the citations between papers; it would be interesting to see how well text similarity measures correlate with citation-based similarity measures.

7 Conclusions

We believe that REFEREE will help to advance the state of the art in recommender systems, especially those which attempt to combine collaborative and content-based similarity information. It provides an open standard and useful tools for the rapid development and evaluation of recommender systems. It also uses metrics that evaluate how the recommender system affects user behavior, which we believe are more useful than the absolute accuracy metrics that have dominated work in the past.

We have developed a prototype hybrid recommender, PD-Live, which uses a novel way of exploiting content-based and collaborative similarity measures. By using content-based similarity to limit the population of documents to recommend and collaborative information to order them, PD-Live has great flexibility and corresponds well to the problem of recommending items to a user in context. The prototype version makes reasonably good recommendations compared to several content-based recommenders, has good performance, and appears to have much potential for improvement.

8 Future work

This paper is the beginning of the future work, which involves getting other researchers interested in recommender systems to participate in the project. In particular, we would like other researchers to develop systems with REFEREE and evaluate its usefulness. We also invite discussion of the use of click-thru and buy-now metrics as an appropriate way to judge recommender systems. Finally, we hope that ResearchIndex becomes a standard testbed for exploring new issues in collaborative filtering. Hybrid recommenders are one such problem; others include recommending in context, and the use of implicit ratings. If you are interested, please email one of the authors.

References

[1] Balabanovíc, M., and Shoham, Y. Fab: Content-based, collaborative recommendation. *Communications of the ACM 40*, 3 (Mar. 1997), 66–72.

[2] BASU, C., HIRSH, H., AND COHEN, W. W. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)* (Menlo Park, July 26–30 1998), AAAI Press, pp. 714–720.

[3] BILLSUS, D., AND PAZZANI, M. J. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning* (1998), Morgan Kaufmann, San Francisco, CA, pp. 46–54.

[4] BOLLACKER, K., LAWRENCE, S., AND GILES, C. L. Discovering relevant scientific literature on the web. *IEEE Intelligent Systems 15*, 2 (March/April 2000), 42–47.

[5] BREESE, J. S., HECKERMAN, D., AND KADIE, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)* (San Francisco, July 24–26 1998), G. F. Cooper and S. Moral, Eds., Morgan Kaufmann, pp. 43–52.

[6] CLAYPOOL, M., GOKHALE, A., MIRANDA, T., MURNIKOV, P., NETES, D., AND SARTIN, M. Combining content-based and collaborative filters in an online newspaper. In *ACM SIGIR Workshop on Recommender Systems* (Berkeley, CA, 1999).

[7] COHEN, W. W. Learning trees and rules with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference* (Menlo Park, Aug. 4–8 1996), AAAI Press / MIT Press, pp. 709–716.

[8] GOLDBERG, D., NICHOLS, D., OKI, B. M., AND TERRY, D. Using collaborative filtering to weave an information tapestry. *Communications of the ACM 35*, 12 (Dec. 1992), 61–70.

[9] GOLDBERG, K., ROEDER, T., GUPTA, D., AND PERKINS, C. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval 4*, 2 (2001), 133–151.

[10] GOOD, N., SCHAFER, J. B., KONSTAN, J. A., BORCHERS, A., SARWAR, B., HERLOCKER, J., AND RIEDL, J. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99)* (Menlo Park, Cal., July 18–22 1999), AAAI/MIT Press, pp. 439–446.

[11] HILL, W., STEAD, L., ROSENSTEIN, M., AND FURNAS, G. Recommending and evaluating choices in a virtual community of use. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems* (1995), pp. 194–201.

[12] HOFMANN, T., AND PUZICHA, J. Latent class models for collaborative filtering. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)* (S.F., July 31–Aug. 6 1999), D. Thomas, Ed., Morgan Kaufmann Publishers, pp. 688–693.

[13] LAWRENCE, S., GILES, C. L., AND BOLLACKER, K. Digital libraries and autonomous citation indexing. *Computer 32*, 6 (June 1999), 67–71.

[14] LEE, W. S. Collaborative learning for recommender systems. In *Proceedings of the Eighteenth International Conference on Machine Learning* (2001), Morgan Kaufmann Publishers, pp. 314–321.

[15] MCJONES, P. Eachmovie collaborative filtering data set. http://www.research.digital.com/SRC/eachmovie, 1997.

[16] PAZZANI, M. J. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review 13*, 5-6 (1999), 393–408.

[17] PENNOCK, D. M., HORVITZ, E., AND GILES, C. L. Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence* (2000), pp. 729–734.

[18] PENNOCK, D. M., HORVITZ, E., LAWRENCE, S., AND GILES, C. L. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)* (SF, CA, June 30– July 3 2000), Morgan Kaufmann Publishers, pp. 473–480.

[19] POPESCUL, A., UNGAR, L. H., PENNOCK, D. M., AND LAWRENCE, S. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *17th Conference on Uncertainty in Artificial Intelligence* (Seattle, Washington, August 2–5 2001), pp. 437–444.

[20] RASHID, A. M., ALBERT, I., COSLEY, D., LAM, S. K., MCNEE, S., KONSTAN, J. A., AND RIEDL, J. Getting to know you: Learning new user preferences in recommender systems. In *Proceedings of the 2002 International Conference on Intelligent User Interfaces* (San Francisco, CA, 2002), pp. 127–134.

[21] RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work* (1994), pp. 175–186.

[22] SARWAR, B. M., KONSTAN, J. A., BORCHERS, A., HERLOCKER, J., MILLER, B., AND RIEDL, J. Using filtering agents to improve prediction quality in the GroupLens research collaborative filtering system. In *Proceedings of ACM CSCW'98 Conference on Computer-Supported Cooperative Work* (1998), pp. 345–354.

[23] SCHEIN, A. I., POPESCUL, A., UNGAR, L. H., AND PENNOCK, D. M. Methods and metrics for cold-start recommendations. To appear in *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002).*

[24] SHARDANAND, U., AND MAES, P. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems* (1995), pp. 210–217.

[25] TURPIN, A. H., AND HERSH, W. Why batch and user evaluations do not give the same results. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval* (New Orleans, LA, September 9–13 2001), pp. 225–231.

[26] UNGAR, L. H., AND FOSTER, D. P. Clustering methods for collaborative filtering. In *AAAI Workshop on Recommendation Systems* (1998).

[27] VOORHEES, E., AND HARMAN, D. Overview of the ninth text retrieval conference (trec-9). In *NIST Special Puclibation 500-249: The Ninth Text REtrieval Conferece (TREC-9)* (Gaithersburg, Maryland, 2001), pp. 1–14.