

# Web Recommender System Implementations in Multiple Flavors: Fast and (Care-)Free for All

Olfa Nasraoui, Zhiyong Zhang, and Esin Saka  
Department of Computer Engineering and Computer Science  
Speed School of Engineering  
University of Louisville  
Louisville, KY 40292

## ABSTRACT

In this paper, we present a systematic framework for a fast and easy implementation and deployment of a recommendation system for one or several Websites, based on any available combination of open source tools that include crawling, indexing, and searching capabilities. The supported recommendation strategies include several popular flavors such as content based filtering (straight forward), collaborative filtering (more complex), rule-based, as well as approaches that deal with meta-content, (non-textual) attributes and ontologies, and other variants that include meta-attributes about the user, such as elaborate user profiles, as well as business strategy rules. The biggest advantage of this approach is that for content-based filtering, it allows client or proxy controlled integration of several websites.

## 1. INTRODUCTION AND MOTIVATIONS:

Recommender systems have been attracting more and more attention as a suitable approach to counteract information overload and help the users of the Web information space find what they need faster. While there have been significant advances in the theoretical and algorithmic aspects of recommender systems, there are no clear and systematic implementation approaches that have been published, as most of these tend to be cloaked under thick walls of corporate secrecy. Also, typical commercial packages may come at a hefty price that may not be affordable to smaller or start-up e-commerce and e-media websites, including the increasing number of purely non-profit community information Websites and forums that cannot even be classified yet as businesses.

In this paper, we present a systematic framework for a fast and easy implementation and deployment of a recommendation system on a Website (or several affiliated or subject-specific websites), based on any available combination of open source tools that include crawling, indexing, and searching capabilities. The supported recommendation strategies include several popular flavors such as content based filtering (straight forward), collaborative filtering (more complex), rule-based, as well as approaches that deal with meta-content (non-textual) attributes and ontologies, and other variants that include meta-attributes about the user, such as user profiles, as well as other business strategy rules. Note that this paper is mainly about ways to *easily "implement" (existing)* recommendation strategies by using a search engine software when it is available, which therefore can be expected to benefit research and real life applications by taking advantage of search engines' *scalable* and *built-in* indexing and query matching features, instead of implementing a strategy from scratch.

### 1.1 Advantages of the Search Engine Tweaking Approach to Recommender System Design

- **Multi-Website Integration by Dynamic Linking:** Easily and seamlessly integrate multiple websites, even those that are hosted on disparate web servers. This facilitate the *dynamic, personalized, and automated linking* of partnering or affiliate websites (e.g. several educational, research, and outreach websites related to a common topic).
- **Giving Control Back to the User or Community instead of the website/business:** The content-based filtering approach (and to some extent the collaborative filtering variations when implemented on a proxy) are no longer confined to the realm of the server, and thus no longer have to serve only the interests of a particular website or business. This means that the user (or a community of users) can install such a system on their own computers (or a proxy machine in case of a community) to unify and dynamically link any group of websites that are relevant to their interests. A group of users (for instance students in the same school district, workers in the same company, or college students in the same university department) can similarly *share* their usage sessions to enrich the collaborative part of the recommendations, if they install the proposed system on a common proxy.
- **The Open Source Edge:** There are currently several available open source implementations for search engines which can form the foundation for several types of recommendation strategies as will be explained in this paper. Being open source, they offer the potential for improvements by the open source community, as well as providing a good, easy, and open platform for teaching and research, with potential to be extended and enhanced by different teams working on Web personalization and Web search.
- **The Information Retrieval Legacy:** Being based on information retrieval, this approach allows inheriting from all previous research and advances in this area to also benefit the field of recommendation systems. This includes inheriting from advances in such areas as: Information Filtering, Personalized Information retrieval, and Query reformulation and expansion.
- **The Semantic Web Vision:** In the same way that different advances have taken place in the field of Information retrieval, significant efforts and developments are already taking place in the area of the *Semantic Web*, where the Web pages' content is handled based on their *meaning, and not just their lexical content*. Developments that can improve the quality of information retrieval by taking semantics into account, can now also have an impact on recommender systems in a seamless manner. For example, *thesauri* will be able to get integrated easily. Thus, the power of Semantic Recommender Systems can be expected to benefit from and evolve with the Semantic Web.

The rest of this paper is organized as follows. In Section 2, we give an overview of Web personalization including recommender systems. In Section 3, we give an overview of the structure and functionalities of a typical search engine implementation. In Section 4, we explain the procedure needed to harness a search engine to implement several types of recommender systems. In Section 5, we present some implementations of the proposed methodologies, and finally make our conclusions in Section 6.

## 2. BACKGROUND ON WEB PERSONALIZATION

In the late 90s, Jeff Bezos, CEO of Amazon once said, If I have 3 million customers on the Web, I should have 3 million stores on the Web [21]. Web personalization tailors a user's interaction with the Web information space based on information gathered about them. The recommendation process follows a decision making process that typically ends up with generating dynamic Web content on the fly, such as adding hyperlinks to the last web page requested by the user, in order to facilitate access to the needed information on a large website [21] [13] [16]. It is usually implemented on the Web server, and relies on data that reflects the users interest implicitly (browsing history as recorded in Web server logs) or explicitly (user profile as entered through a registration form or questionnaire). The *implicit* approach is the focus of the work presented in this paper. Personalization can be used to achieve several goals, ranging from increasing customer retention and loyalty on e-commerce sites [21] to enabling better search by making results of Web information retrieval/search more aware of the context and user interests [10]. In addition, personalization can help convert browsers into buyers, increase cross-sell by recommending items related to the ones being considered, improve web site design and usability, help visitors to quickly find relevant information on a large website, and more generally help businesses maintain a more effective Customer Relationship Management (CRM) [7]. Recommender systems have already found some success in real e-commerce applications such as Amazon [12] and CDNow [1], where they are used to recommend to online shoppers, products and services that they might otherwise never discover on their own. There have also been several pioneering research system prototypes, such as Syskill and Webert [18], PHOAKS [24], Fab [3], and GroupLens [11] [20]. Recommender systems can be classified in a variety of ways [21] [17] [4] depending on their inputs and on the recommendation algorithm used, including:

- Content-based or Item-based filtering: Content-based filtering systems recommend items to a given user, which are deemed to be similar to the items that the same user liked in the past. Item similarity is typically based on domain specific item attributes (such as author and subject for book items, artist and genre for music items). Classical examples include Syskill and Webert [18], and Fab [3].
- Collaborative filtering: Based on the assumption that users with similar past behaviors (rating, browsing, or purchase history) have similar interests, a collaborative filtering system recommends items that are liked by other users with similar interests [21]. This approach relies on a historic record of all user interests such as can be inferred from their ratings of the items on a website (products or web pages). Rating can be explicit (explicit ratings, previous purchases, customer satisfaction questionnaires) or implicit (browsing activity on a website or clickstreams). Typical examples include GroupLens [11][20]. Computing recommendations can be based on lazy or eager learning to model the user interests. In lazy learning all previous user activities are simply stored, until recommendation time, when a new user is compared against all previous

users to identify those who are similar, and in turn generate recommended items that are part of these similar users interests. On the other hand, eager learning relies on data mining techniques to learn a summarized model of user interests (a decision tree, clusters/profiles, etc) that typically requires only a small fraction of the memory needed in lazy approaches. One particular eager modeling approach, that has lately received increasing attention, is known as *Web Usage Mining* [15][23][22]. *Web Usage Mining* consists of applying machine learning or data mining techniques to discover interesting usage patterns and statistical correlations between web pages and user groups. This learning frequently results in automatic user profiling, and is typically applied offline, so that it does not add a burden on the web server. Collaborative filtering methods are completely independent of the intrinsic properties of the items being rated or recommended. In particular, items that may be hard to describe using attributes, such as video, audio and images, as well as semantically rich text data can still be recommended based on latent similarities that are only captured through the *social* process of collaborative filtering, hence, often suggesting completely *new* types of items that are *different* from, and yet *associated* with previously rated items.

- Knowledge Engineering or Rule-based filtering: In this approach, used frequently to customize products on e-commerce sites such as Dell on Line, the user answers several questions, until receiving a customized result such as a list of products.
- Demographic recommender systems: In this approach, items are recommended to users based on their demographic attributes. The recommendations can be based on handcrafted stereotypes derived from marketing research or on machine learning techniques [17] that learn to predict users' preferences from their demographic attributes.
- Hybrids: Each recommendation strategy has its own strengths and weaknesses. Hence, combining several recommendation strategies can be expected to provide better results than either strategy alone [17][4]. Most hybrids work by combining several input data sources or several recommendation strategies.

## 3. OPEN SOURCE SEARCH ENGINES - THE CASE OF "NUTCH"

Before we present our systematic procedure to make a search engine deliver recommendations, we will briefly explain the architecture of Web search engines. Most search engines rely on four crucial components that perform the following functions usually in this order: *(i) Crawling*: A crawler retrieves the web pages that are to be included in a searchable collection, *(ii) Parsing*: The crawled documents are parsed to extract the terms that they contain, *(iii) Indexing*: A *reverse index* is typically built that maps each parsed term to a set of pages where the term is contained, *(iv) Query matching*: After the index has been built, input queries in the form of a set of terms can be submitted to a search engine interface or to a query matching module that compares this query against the existing index, to produce a ranked list of results or web pages. Below, we focus on two open source products that enable a fast and free implementation of Web search, first a text search engine library called *Lucene*, and then the Web search engine, *Nutch*, that is built on the Lucene text search library.

### 3.1 Lucene

#### 3.1.1 General overview and special features

*Apache Lucene* [6] is an open source project of the Apache

Software Foundation.<sup>1</sup> *Lucene*<sup>2</sup> is a high-performance, full-featured *text search engine library* written in Java, and can support any application that requires full-text search, especially cross-platform. Examples of using Lucene include *Inktomi* and *Wikipedia*'s search feature. Lucene offers powerful features through a simple API, including scalable, high-performance indexing, as well as powerful, accurate and efficient search algorithms. It is also available as Open Source software under the Apache License which allows using Lucene in both commercial and Open Source programs. Of particular interest to this paper are the flexible and powerful features of Lucene's search algorithms<sup>3</sup>: ranked searching; powerful query types: phrase, wildcard, proximity, fuzzy, range, and more; fielded searching (e.g., title, author, contents), date-range searching, sorting by any field, multiple-index searching with merged results, and allowing simultaneous update and searching.

### 3.1.2 Query Syntax

A query in Lucene is broken up into *terms* and *operators*, with two types of terms: *Single Terms* and *Phrases*. A *Single Term* is a single word such as "test" or "hello". A *Phrase* is a group of words surrounded by double quotes such as "hello dolly". Multiple terms can be combined together with *Boolean operators* to form a more complex query. Lucene also supports *fielded data*. A search can thus be performed by specifying a field, or by using the default field. The field names and default field are implementation specific. Lucene queries can also *boost* a particular term so that it weighs more heavily in the matching process, for example the query "jakarta^4 apache" will cause documents containing the term "jakarta" to be considered more relevant compared to the ones containing the term "apache". Note that the boosting and (default) Boolean OR based queries can easily be used to form *vector space queries* (the term frequencies become the boosting weights)<sup>4</sup> or by sorting the results of a Boolean OR query based on vector space similarity, as described for example in the TREC & HARD' 05 record in [8][9].

### 3.1.3 Scoring

Given an input query, Lucene scores the results based on the TF-IDF measure where the documents are represented in the vector space model.

## 3.2 Nutch

Nutch (<http://lucene.apache.org/nutch/>) is an open source *Web search software*, built on Lucene Java, that adds web-specifics, such as a *crawler*, a *link-graph database*, *parsers for HTML* and *other document formats* (such as pdf, Microsoft Powerpoint and Word, plain text, etc). Other document types can be parsed by programming their specific *plugin*. Nutch maintains a Database of pages and links. For each web page that is indexed, only the *URL* and *title* are stored, while the *anchor* and the *content* are only indexed. A *document* is a sequence of *Fields*. A *field* is a  $\langle name, value \rangle$  pair, where *Name* is the name of the field, e.g., title, body, subject, date, etc; and *Value* is text. Field values may be stored, indexed, analyzed (to convert to tokens), or vectored. Lucene's index is an *Inverted Index* that maps a term to a *field ID*, and a set of document IDs, with the position within each document. Nutch uses Lucene search which supports both *primitive queries* (term, phrase, and Boolean) and *derived queries* (prefix and/or wildcard). Given a query, Nutch by default searches *URLs*, *anchors*, and *content* of documents, and also rewards for *proximity* of the query words in the documents. Finally, we note that Nutch also uses *N-Grams* to index very common terms with their neighbors, which in turn improves *phrase search*.

<sup>1</sup><http://www.apache.org>

<sup>2</sup><http://lucene.apache.org/>

<sup>3</sup><http://lucene.apache.org/java/docs/features.html>

<sup>4</sup><http://www.cs.virginia.edu/~xj3a/research/TREC/index.htm>

## 4. PROPOSED METHODOLOGY

### 4.1 Requirements for tweaking a search engine to work like a recommender system

There are two principal requirements to being able to use a search engine to provide recommendations:

- **An index:** The source of the recommendations must be stored or rather indexed in a format that is easy to search.
- **A querying mechanism:** Regardless of the recommendation strategy and regardless of what has been indexed, the *input* to the recommendation procedure *must be transformable into a query* that is expressed in terms of the entities upon which the index is based. For instance if the index maps every keyword in a given vocabulary to a set of pages that contain this keyword, then the query needs to be expressed in terms of these keywords.

Hence, there appear to be two components that can define how the search engine will deliver recommendations: the *index* and the *query*. In order to realize a given recommendation strategy, it is therefore necessary to at least define how each of these components will be handled. In the following subsections, we assume that both a raw session and a simple collaborative profile consist of a *set* of indexed items/Web pages  $P_i$  (note that this set can also be considered as a *vector* in the space defined by all possible indexed items), thus

$$\text{raw-session} = \{P_1, P_2, \dots, \text{etc}\}$$

$$\text{profile} = \{P'_1, P'_2, \dots, \text{etc}\}$$

### 4.2 Methodologies for tweaking a search engine to work like a recommender system

#### 4.2.1 Content-based filtering

The content-based filtering approach is straightforward in case the recommended items are HTML or text-based pages. The idea behind content-based filtering is that given a few pages that a user has viewed, the system recommends other pages with content that is similar to the content of the viewed pages. Most content-based filtering systems store an item-to-attribute matrix that describes every item as a feature vector with several attributes. In the case of textual content, the bag of words is a simple and popular approach to representing the items (in this case web pages). Given a set of items that have been viewed or rated by a user, the recommendation system will then compute the similarity between the rated items and all other items to determine which items are most similar to what the user showed interest in, and then recommends the top items. However, a search engine works by reverse-indexing the items (in this case, web pages), and then comparing a submitted query to the indexed items in order to determine the closest items. For our purpose, the recommendation process amounts to performing the following two steps:

#### 1. Preliminary Crawling and Indexing (*done offline*):

The first step will consist of crawling the website(s) that will contribute to the *content* of the recommendations, and then forming a reverse index that maps each keyword to a set of pages in which it is contained. Since we plan to compare a query to a set of web pages in *vector space* format, we store the most frequent terms in each document as a *vector field*, that is indexed and used later in retrieval by submitting a fielded query.

#### 2. Query Formation and Scoring:

This step essentially consists of transforming a new user session into a query that can be understood by the search engine. First, each URL in the user session is mapped to a set of content terms that are most characteristic of this URL (top  $k$  frequent terms) using an added package `net.nutch.searcher.page`

Then these terms are combined with their frequencies to form a query vector, that gets submitted to the Nutch search interface as a *Fielded* query, i.e. the query vector is compared to the indexed Web document vector field. Finally, the results are ranked according to the cosine similarity of their content with the query vector in the vector space domain (introduced as a modification of the default scoring mechanism of `SortComparatorSource` in the `LuceneQueryOptimizer` class which is part of the package `net.nutch.searcher`). We are currently also in the process of experimenting with a Boosted disjunctive Boolean query<sup>5</sup> as discussed within the context of information retrieval in [8][9].

session  $\rightarrow$  URLs  $\rightarrow$  terms  $\rightarrow$  fielded query vector  $\rightarrow$  results (ranked according to cosine similarity (result vector, query vector))

In the *final query formation*, in order to give preference to the most recent context (i.e. web page requests), the content vectors of the pages in the session are weighted differently, so that more recent pages receive higher weights than older pages. This has the effect of *forgetting* the content of older pages in the session compared to the more recent requests. This process is shown in Fig. 1.

Once the query has been formed, it is delivered to the search engine for further processing, and the search engine returns a ranked set of items/web pages that are sorted by the *cosine similarity* between the *vector of TF-IDF-weighted* terms of the query (which in turn represents a user’s session) and the *TF-IDF weighted* terms of the indexed pages. Hence the results accomplish the goal of content-based filtering. The user session is transformed into a query that accumulates all the term weights of the pages that are visited. However, the weight  $w_1$  of an older page  $P_1$  gets weakened with the arrival of each new page in the session by multiplying it by a forgetting factor  $F$  in  $(0,1)$  as follows:  $w_1 \leftarrow F.w_1$ . The *cumulative session to content query transformation* is illustrated in Fig. 2. Note that some essential features of text retrieval, used as part of *Nutch*, such as *TF-IDF* are inherited in the proposed content-based filtering method. This is desirable as we do not want the very popular terms in a Web collection to dominate the matching process. For example, a website or a set of affiliated websites that are devoted to the topic “Solar Astronomy” are expected to contain a large number of Web pages containing either one of these terms. Thus, these terms may not be useful in distinguishing between the different pages. The *TF-IDF* measure will suppress the contribution of such common or popular terms exactly as desired, and focus instead on other terms that help better distinguish between the pages.

#### 4.2.1.1 Anchor and URL matching: .

An option supported by most search engines would be to include the anchor information of the viewed web pages (the anchor text is the text used in hyperlinks that point to a given page) when formulating the queries. Here, once the anchor data is added to the query vector (in the same way as the actual content terms), they are submitted as an (anchor+content) query, and this query may be matched against the content index or the content and anchor index. Most search engines end up comparing a given query not only to the content of the indexed Web pages, but also to their anchors, and even their URLs. For instance, given a query, Nutch by default searches *URLs, anchors, and content* of documents, and also rewards for *proximity* of the query words within the documents. We consider this to be a powerful feature for Content-based filtering recommendations, since the item matching is not only based on the intrinsic content, and takes into account proximity of the terms, but it is actually extended to the “location label” or URL of the item and its position in a website hierarchy and even rewards similarly named items in different areas of the

<sup>5</sup><http://www.cs.virginia.edu/~xj3a/research/TREC/index.htm>

website. Anchors can also play an important role in *enriching* and to some extent, *disambiguating* intrinsic content that is *lexically* similar. We see this as an added bonus of the search engine based realization of a recommendation system, since the anchor and URL indexing and matching features are a natural component that is taken for granted in many search engines, and yet could be considered a hassle to implement from scratch as part of a recommendation approach.

Figure 1: Cumulative session to content query transformation in the Content-based Filtering approach

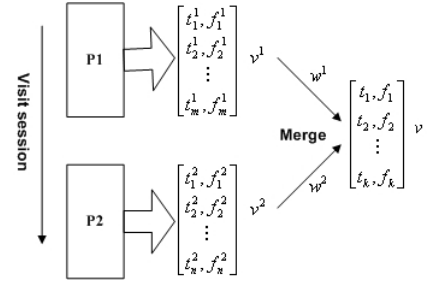
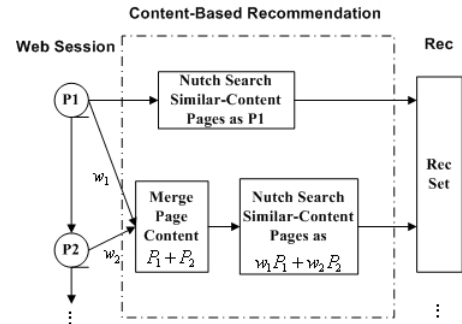


Figure 2: Content-based Filtering implemented via a Search Engine



#### 4.2.2 Collaborative filtering

Compared to content-based filtering, the collaborative filtering approach is more tricky since it involves multiple users and forming neighborhoods of these users, however there are two ways that a crawl-index-search engine can be tweaked to deliver collaborative filtering recommendations:

##### 4.2.2.1 Full-Fledged Collaborative Filtering.

A possibly complex approach would involve first creating an index consisting of the user sessions (instead of web pages) in the URL domain (instead of the keyword domain), then using this index as the basis for query matching (again in the URL domain), thus submitting a query in the form of a set of URL IDs instead of a set of words, and retrieving as result, a set of similar sessions. Finally, the frequency of occurrence of the pages (URLs) contained in all the retrieved similar sessions are accumulated, and the pages/URLs are sorted in decreasing order of their relevance. This forms the final result. This final operation needs to be programmed as a plugin to the query matching component. Note that the query transformation component of this search engine tweak is straightforward, i.e.:

session = query

Also, this methodology amounts to pure collaborative filtering, where the content of the pages plays no role in the recommendations, and *only* the similarities between the user activities drive the recommendations. From a search engine tweaking point of view, it is important to note that this approach puts its entire leverage on *the indexing component* and not the query formation, since the sessions are indexed as if they were documents, and hence the crawling component becomes obsolete. The only difference compared to content-based filtering is that the *index* is in the *URL domain* instead of the keyword domain. That is, that the query matching relies on a vector space representation of the sessions in the URL domain instead of a vector space representation of the documents in the keyword domain.

As a legacy from the TF-IDF scoring approach used in scoring, our collaborative filtering approach will inherit its suppression of very popular URLs that are visited frequently in most sessions, indeed a very desirable side-effect. This is because most popular pages tend to include navigational pages that tend to be located at the crossroads of many paths through the website (and which tend to be excessively linked to on the website).

#### 4.2.2.2 Hybrid Content via Collaborative Filtering with Cascaded/Feature Augmentation Combination.

Hybrid Content via Collaborative Filtering amounts to performing the following steps:

1. **Preliminary Crawling and Indexing (*done offline*):**

This preliminary step will consist of crawling the website(s) that will contribute to the *content* of the recommendations as in content-based filtering, and then forming a reverse index that maps each keyword to a set of pages in which it is contained. This step is identical to Step 1 in 4.2.1. This index is called *Index 1* (in term space).

2. **Preliminary Indexing of Prior User Sessions using Method described in Full-Fledged Collaborative Filtering in Sec. 4.2.2.1 (*done offline*):**

Create an index consisting of the user sessions (instead of web pages) in the URL domain (instead of the keyword domain), this will allow matching a new user session in the URL domain to similar user sessions in the URL domain. We call this index *Index 2* (in URL space).

3. **Matching New User Session to Previous Sessions to Form a Collaborative Session:**

The next step is to match a new user session to the existing sessions in *Index 2* in order to retrieve the top  $N$  similar sessions. The frequency of occurrence of the pages (URLs) contained in all the retrieved similar sessions are accumulated, and the pages/URLs are sorted in decreasing order of their relevance. This forms an *intermediate* result that is identical to the final result of *Full-Fledged Collaborative Filtering*. We call this result a collaborative session, because it has the same structure as the original raw session, and will further go through content enrichment in the next step.

4. **Query Formation and Scoring:**

The final step proceeds similarly to Content-based filtering as explained above. That is, first we transform this collaborative-session into a content query by merging the Web pages' content as in content-based filtering, and then submit this content query to the search engine that matches it against its content index, *Index 1*.

The entire transformation process can be summarized as follows:

session (submit to *Index 2* in URL space) → similar sessions → collaborative-session → URLs → terms →

fielded query vector (to submit to *Index 1* in term space) → results (ranked according to cosine similarity (result vector, query vector))

Note that this approach will result in a *hybrid (collaborative and content)* recommender system, since the final recommendations will also take content into consideration. The *collaborative* ingredient comes from the fact that the content is not only formed from the current session, but also combines the content from a neighborhood of “similar” sessions. Furthermore, this approach may be classified into the family of *cascaded hybrid recommendation strategies* according to [4], as well as *feature augmentation hybrid recommendations*, also discussed in [4]. The latter was defined as “*methods where the output from one technique is used as an input feature to another*”, which is what is performed in this case. At the same time, *cascaded* hybrids were defined in [4] as “*One recommender refines the recommendations given by another*”, which is also the case of the described approach because *both* stages result in a set of URLs (thus recommendations).

We call this approach *content-via-collaborative-filtering*, and it is illustrated in Fig. 3. Compared to *pure Content-based Filtering*, this approach *first modifies the session* by including items (pages) of interest to *similar* users, and then performs content-based filtering on the resulting session. This approach is reminiscent of Pazzani’s *Collaborative-via-Content* recommendation approach [17], however the latter performs the two steps in *reverse* order, i.e. Content-based filtering, followed by Collaborative filtering. When it comes to the *final query formation*, the only difference between the combined approach and the pure Content-based Filtering approach is that there is *no forgetting* of the content of older pages in the session compared to the more recent requests, though this feature can be accomplished easily in Step 3 if *older* pages are assigned lower weights.

#### 4.2.2.3 Hybrid Content via Profile-based Collaborative Filtering with Cascaded/Feature Augmentation Combination.

Hybrid Content via Profile-based Collaborative Filtering amounts to performing the following four steps:

1. **Preliminary Mining of Usage Profiles (*done offline or online with scalable Web usage mining techniques*):**

A simpler profile based approach assumes that we have already mined collaborative user profiles [5, ?, ?], or possibly that we have been mining usage micro-clusters incrementally, for example in a streaming framework [14]. Once a set of collaborative user profiles is available, they will form the basis for collaborative filtering.

2. **Preliminary Crawling and Indexing (*done offline*):**

Another preliminary step will consist of crawling the website(s) that will contribute to the *content* of the recommendations. This step is identical to Step 1 in Sec. 4.2.1.

3. **Matching New User Session to Previous Profiles to Form a Collaborative Session:**

The next step is to match a new user session to the existing profiles in order to determine close (or activated) profiles, and then combine the URLs in these profiles based on the degree of similarity between the new session and each originating profile to form a new collaborative-session,

4. **Query Formation and Scoring:**

The final step transforms the collaborative-session into a content query as in the last step of Content-based filtering explained above.

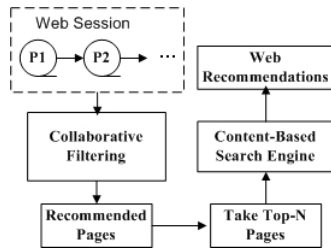
The entire transformation process can be summarized as follows:

session → close-profiles → collaborative-session → URLs → terms → query → results → re-rank results according to cosine similarity (result vector, query vector)

Note that just as in the case of *content-via-collaborative-filtering* described in Sec. 4.2.2.2, this approach will result in a *hybrid (collaborative and content)* recommender system, since the final recommendations will also take content into consideration. The collaborative ingredient comes from the fact that the content is not only formed from the current session, but also combines the content from a neighborhood of “similar” sessions. Also, this approach may be classified into the family of *cascaded hybrid recommendation strategies*, as well as *feature augmentation hybrid recommendations* [4].

We call this approach *content-based-via-collaborative-profile-filtering*, which is similar to [17], but it relies on profiles as a summary of all the previous sessions. This approach is also illustrated in Fig. 3.

**Figure 3: Cascaded Content-based-via-collaborative-profile-filtering implemented via a Search Engine**



#### 4.2.2.4 Hybrid Content and Profile-based Collaborative Filtering with Weighted Combination.

The previously described Hybrid Content via Profile-based Collaborative Filtering works by *cascading* the collaborative filtering recommendations onto the content-based filtering to produce the final recommendations. Hence it works in *stages*. An obvious alternative would be to combine the collaborative filtering and content-based filtering *in parallel* instead of sequential manner. This approach falls within the family of *weighted hybrid recommendation strategies*, defined in [4] as methods where “the scores (or votes) of several recommendation techniques are combined together to produce a single recommendation”. We call this approach *Weighted content-based-and-collaborative-profile-filtering*, and it is illustrated in Fig. 4. The recommendation procedure is performed using the following five steps:

1. **Preliminary Mining of Usage Profiles (done offline or online with scalable Web usage mining techniques):** This step is identical to Step 1 in Sec. 4.2.2.3.
2. **Preliminary Crawling and Indexing of the Content (done offline):** This step is identical to Step 1 in Sec. 4.2.1.
3. **Matching New User Session to Previous Profiles to Form a Collaborative Session:** The next step is to match a new user session to the existing profiles in order to determine close (or activated) profiles, and then combine the URLs in these profiles based on the degree of similarity between the new session and each originating profile to form a new collaborative-session. This collaborative session with its inherent ranking of the URLs is then treated like a first component of the final recommendations (Recommended Set 1). the overall transformation process so far is as follows: **session** → **URLS** → **closest profiles** → **Recommended Set 1**
4. **Query Formation and Scoring:** this steps essentially consists of transforming the same new user session into a

query that can be understood by the search engine, exactly as in pure content-based filtering. First, each URL in the user session is mapped to a set of terms that are most characteristic of this URLs, and then these terms are combined to form a query as in Fig. 1, i.e:

**session** → **URLS** → **terms** → **query** → **Recommended Set 2**

#### 5. Final Collaborative and Content-based Recommendation Combination:

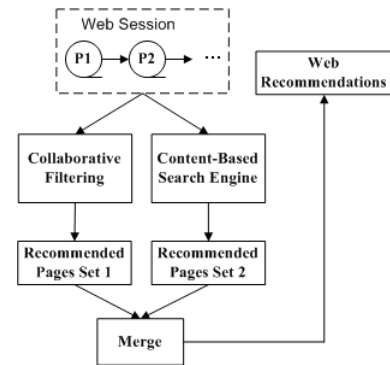
The final step consists of combining the two above recommended sets of items into a coherent list of recommendations. The main idea is to merge the two lists and sort them by the matching score assigned to each item. This transformation process can be denoted as follows:

**Recommended Set 1** ⊕ **Recommended Set 2** → **Final Recommended Set**

Note that we do not discuss the details of how the merging takes place, since many possible strategies can be implemented to do this. For example, one possible variation is to *weight* the scores of the recommendations in the two sets differently before sorting them into a final list, to give a bias toward the collaborative filtering component or the content-based filtering component depending on the application domain and on the performance of the recommendations.

Note that this approach will also result in a *hybrid (collaborative and content)* recommender system, since the final recommendations will take both similar users’ interest (profiles) and content into consideration.

**Figure 4: Weighted Content-based-and-collaborative-profile-filtering implemented via a Search Engine**



#### 4.2.3 Other Recommendation Approaches

The implementation of various other recommendation strategies can also be designed, although at first, the tweaking may not be as obvious. For example, it may be possible to tweak *rule-based, meta-content-attributes*, which are typically non-textual attributes that can be defined by use of *ontologies* (consider movie attributes for instance), user profile attributes such as demographic or declarative preferences, and *external business rules*, etc. Some of these different recommendation options are outlined below.

We can distinguish the following different cases where additional content or user profile attributes need to be incorporated into the recommendations:

**Case 1: The content attribute is stored in a Database** in the form of symbols, codes, etc (for instance on an e-commerce website this may include the *color* of a skirt such as *pink, blue, or black*, or the *type of heel* on a shoe such as *stiletto, platform or flat heel*). In order to handle such attributes, the

code names can be added to the vocabulary of keywords, and may be treated as such in the item or underlying web page description. In this case, the procedure can be easily reduced to that of Content-based filtering as explained above. That is the viewed (or purchased) items can be mapped to a query and submitted to the search engine to return the most matching results. In cases where the attributes are tightly coupled with a specific item (for example the heel type is only attributable to a shoe and not a belt), then the session may be transformed into several separate queries that are submitted to the search engine sequentially, and the results combined. Note that this approach presupposes that the item Database has previously been converted to text and that its attributes have been stored into a reverse index that can make looking up a specific attribute easy and fast at recommendation time. An alternative to text transformation can be accomplished by storing the attributes as *fields* which are also indexable and searchable by Lucene (see Sec. 3.1.2).

**Case 2: The content attributes are described or connected via an ontology or a set of rules:** In this case, we can modify the *similarity* measure that is used by the search engine to match a query to existing content. Basically, the similarity measure should reflect all the relevant rules and ontological relationships between the different attributes and items to be compared in order to provide the recommendations. Note that here again, Lucene’s ability to index and search *fielded* data can help deal with external attributes and simple rules (see Sec. 3.1.2).

**Case 3: An elaborate user profile** is available with non-textual attributes that describe a user, such as demographic attributes (age, income, gender, marital status, etc), or how a customer relates to certain known customer segments, such as (heavy spender, loyal customer, new customer, vulnerable customer, etc). This case can be solved by modifying the *similarity* matching so that all the rules that relate the recommended item contents to the user attributes are taken into account when performing the matching.

**Case 4: A set of business rules** are to be taken into account to recommend the items, which correspond to neither content-based nor collaborative filtering. This case is similar to the previous case in that the *similarity* matching module needs to be modified to implement these rules, so that a similarity degree may be suppressed or promoted depending on how a recommendation fits a given rule.

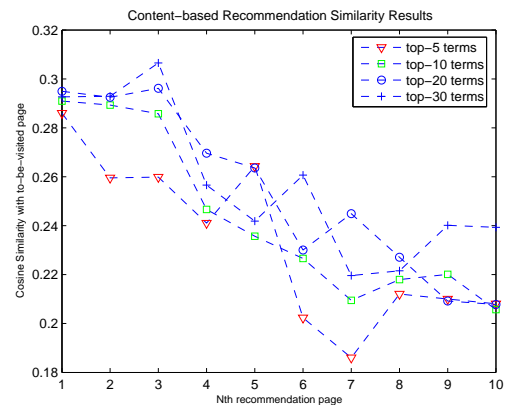
## 5. IMPLEMENTATION EXAMPLES

Our implementation started by crawling the web pages in the louisville.edu domain, where we limited the crawling to a depth of 6 levels resulting in 30,405 pages (this corresponds to Step 1 of content-based filtering). The content was crawled and indexed using nutch (23,097 unique terms were indexed), and the nutch search engine application was launched to accept queries and match them on the fly to the indexed content as they arrive. A proxy was set at one port on our server <http://136.165.45.122:8089/> based on the Open Source SQUID Web proxy software (<http://www.squid-cache.org/>) and additional C code to track each session, convert it to an appropriate query, and submit this query to the nutch open source search engine to generate recommendations. Thus in order to view the recommendations which are available as the user navigates any of the web pages that have been crawled, the user must first set their browser to use <http://136.165.45.122:8089/> as a proxy, and then continue their normal navigation in the web pages located within the domain of interest (in our case this was louisville.edu), where the *recommendations* link will be shown at the top of each page. The proxy approach offers a fast and non-invasive approach to add recommendations as was done in [2].

We are currently making progress on several recommendation strategies, including collaborative filtering methods, but have so far completed the simplest strategies which are the

content-based filtering and anchor-based recommendations (all anchor terms of the sessions’ browsed pages are merged into a Boolean query, and matched against the reverse index). In order to extract the content of a given URL, we used Nutch’s built in functionalities for parsing various types of file extensions, and added a plugin for stop word elimination. The quality of *content-based filtering* recommendations was assessed by comparing the contents of the pages in the recommendation set to the pages to be visited in a real session, using the cosine similarity. The *k* most frequent terms in a page were used to form its vector space representation, First, we varied *k* to take the values: 5, 10, 20, 30, 40, and show the quality in Fig. 5. We noticed that the quality of the results saturated around *k* = 30, and thus do not show results for *k* > 30. The results used for evaluation purposes 450 sessions consisting of at least 3 clicks, and chosen randomly from one day’s web logs. Then we fixed the maximum number of terms (*k* = 10) in the vector space representation and determined the quality for sessions of different lengths in Fig. 6.

**Figure 5: Preliminary Results for Content-based filtering implemented via a Search Engine: Cosine similarity between Nth recommended page and to-be-visited pages for different maximum number of terms (*k*) used in vector space representation (forgetting factor  $F=0.5$ )**



**Figure 6: Preliminary Results for Content-based filtering implemented via a Search Engine: Cosine similarity between Nth recommended page and to-be-visited pages for different lengths of user sessions, and fixed maximum number of terms (*k* = 10) used in vector space representation (forgetting factor  $F=0.5$ )**

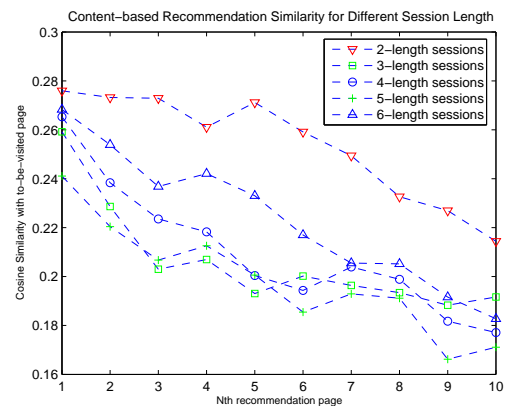


Table 1 shows that the time required for the content-based filtering recommendations, on a Linux server with Intel Xeon

**Table 1: Average time to Recommendations for Content-Based Filtering vs. Number of Top Frequent Terms used in the Vector Space Representation**

$k$	5	10	20	30	40
time in seconds	1.6	2.08	2.8	3.6	4.2

2.80GHz, 2CPU, and 2G memory, is modest (about 2 seconds for  $k = 10$  top frequent terms used in the vector space representation), and that it grows almost linearly with the number of terms used in the vector space representation of the content ( $k$ ).

## 6. CONCLUSIONS

In this paper, we have presented a systematic framework for a fast and easy implementation and deployment of a several types of recommendation strategies on one or more Websites, based on any available combination of open source tools that include crawling, indexing, and searching capabilities. There are many advantages of the Search Engine Tweaking Approach to Recommender System Design, which include: (i) the easy and *seamless integration* of multiple websites, such as affiliated e-commerce sites, even those that are hosted on disconnected web servers, (ii) *putting the user in control*: The proposed recommendation implementations are no longer confined to serve the interests of a particular website or business. In fact, the user (or a community of users) can install such a system on their own computers (or proxy in case of a community) to unify and dynamically link any group of websites that are relevant to their interests, (iii) the *Open Source advantage*: that offers the potential for improvements by the open source community, as well as providing an open platform for teaching and research, (iv) Inheriting from the *Information Retrieval legacy*, including all the developments in Information Filtering, Personalized Information retrieval, and Query reformulation and expansion; (v) the *Semantic Web Vision*: Advances in information retrieval based on semantics can now also have an impact on recommender systems in a seamless manner.

### 6.1 Acknowledgments

This work is partially supported by National Science Foundation CAREER Award IIS-0133948 to O. Nasraoui.

## 7. REFERENCES

- [1] Cdnnow.com <http://www.cdnnow.com>.
- [2] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments*, 1995.
- [3] M. Balabanovic. An adaptive web page recommendation service. In *First International Conference on Autonomous Agents, Marinadel Rey, CA, 378-385*, 1997.
- [4] R. Burke. Hybrid recommender systems: Survey and experiments. In *In User Modeling and User-Adapted Interaction, 12(4):331-370*, 2002.
- [5] R. Cooley, B. Mobasher, and J. Srivastava. Web mining: Information and pattern discovery on the world wide web. In *Proc. IEEE Intl. Conf. Tools with AI, Newport Beach, CA, pp. 558-567*, 1997.
- [6] D. Cutting and J. Pedersen. Space optimizations for total ranking, riao (computer assisted ir) 1997. In <http://lucene.sf.net/papers/riao97.ps>.
- [7] P. V. der Putten, J. N. Kok, and A. Gupta. Why the information explosion can be bad for data mining and how data fusion provides a way out. In *In Proc. of the 2nd SIAM International Conference on Data Mining*, 2002.
- [8] X. Jin, J. C. French, and J. Michel. Saic & university of virginia at trec 2005: Hard track. In *In Proceedings of TREC 2005. On-line at <http://trec.nist.gov>*.
- [9] X. Jin, J. C. French, and J. Michel. Saic & university of virginia at trec 2005: Hard track. In *Proc. 14th Text REtrieval Conference (TREC 2005), Gaithersburg, MD, November 15-18, 2005*.
- [10] T. Joachims. Optimizing search engines using clickthrough data. In *In Proc. of the 8th ACM SIGKDD Conference, 133-142*, 2002.
- [11] J. Konstan, B. M. J. Maltz, G. Herlocker, and J. Riedl. Grouplens: Collaborative filtering for usenet news. In *Communications of the ACM, March, p. 77-87*, 1997.
- [12] G. Linden, B. Smith, and J. York. Amazon.com recommendations item-to-item collaborative filtering. In *IEEE Internet Computing*, volume 7.
- [13] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *ACM Workshop on Web information and data management, Atlanta, GA, Nov 2001*.
- [14] O. Nasraoui, C. Cardona, C. Rojas, and F. Gonzalez. Mining evolving user profiles in noisy web clickstream data with a scalable immune system clustering algorithm. In *in Proc. of WebKDD 2003, Washington DC, Aug 2003*.
- [15] O. Nasraoui, R. Krishnapuram, and A. Joshi. Mining web access logs using a relational clustering algorithm based on a robust estimator. In *8th International World Wide Web Conference, Toronto, pp. 40-41*, 1999.
- [16] O. Nasraoui, R. Krishnapuram, A. Joshi, and T. Kamdar. Automatic web user profiling and personalization using robust fuzzy relational clustering. In *in E-Commerce and Intelligent Methods in the series Studies in Fuzziness and Soft Computing, J.Segovia, P. Szczepaniak, and M. Niedzwiedzinski, Ed, Springer-Verlag*, 2002.
- [17] M. Pazzani. A framework for collaborative. In *Content-Based and Demographic Filtering, AI Review, 13(5-6):393-408*, 1999.
- [18] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. In *Machine Learning, 27:313331*, 1997.
- [19] M. Perkowski and O. Etzioni. Adaptive web sites: Automatically learning for user access patterns. In *Proc. 6th int. WWW conference*, 1997.
- [20] B. M. Sarwar, J. A. Konstan, A. Borchers, J. Herlocker, B. Miller, and J. Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *In Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work, Seattle, Washington*, 1998.
- [21] J. Schafer, J. Konstan, and J. Reidel. Recommender systems in e-commerce. In *Proc. ACM Conf. E-commerce*, 1999.
- [22] M. Spiliopoulou and L. C. Faulstich. Wum:a web utilization miner. In *in Proceedings of EDBT workshop WebDB98, Valencia, Spain*, 1999.
- [23] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. In *SIGKDD Explorations*, volume 1, Jan 2000.
- [24] L. Terveen, W. Hill, B. Amento, D. McDonald, and J. Creter. Phoaks: A system for sharing recommendations. In *Communications of the ACM, 40(3), 59-62*, 1997.