Testing and Evaluating Tag Recommenders in a Live System

Robert Jäschke, Folke Eisterlehner, Andreas Hotho, and Gerd Stumme

Knowledge & Data Engineering Group
University of Kassel
Wilhelmshöher Allee 73
34121 Kassel, Germany

http://www.kde.cs.uni-kassel.de/

Abstract

The challenge to provide tag recommendations for collaborative tagging systems has attracted quite some attention of researchers lately. However, most research focused on evaluation and development of appropriate methods rather than tackling the practical challenges of how to integrate recommendation methods into real tagging systems, record and evaluate their performance.

In this paper we describe the tag recommendation framework we developed for our social bookmark and publication sharing system BibSonomy. With the intention to develop, test, and evaluate recommendation algorithms and supporting cooperation with researchers, we designed the framework to be easily extensible, open for a variety of methods, and usable independent from BibSonomy. Furthermore, this paper presents an evaluation of two exemplarily deployed recommendation methods, demonstrating the power of the framework.

1 Introduction

Collaborative tagging systems are web based systems that allow users to assign keywords – so called tags – to arbitrary resources. Tags are used for navigation, finding resources and serendipitous browsing and thus provide an immediate benefit for users. These systems usually include tag recommendation mechanisms easing the process of finding good tags for a resource.

Recommending tags can serve various purposes, such as: increasing the chances of getting a resource annotated, reminding a user what a resource is about and consolidating the vocabulary across the users. Furthermore, as Sood et al. [Sood et al., 2007] point out, tag recommendations “fundamentally change the tagging process from generation to recognition” which requires less cognitive effort and time.

Our contributions with this paper are: (i) presenting and evaluating a tag recommendation framework deployed in BibSonomy, an open collaborative tagging system, (ii) providing researchers a testbed to test and evaluate their methods in a live system, and (iii) showing first results which indicate the power of the framework to improve recommendation performance by clever selection strategies.

This paper is structured as follows: In Section 2 we introduce BibSonomy and motivate the task of tag recommendations; in Section 3 we review related work in the field and continue in Sec. 4 to explain the details of our tag recommendation framework. Then we elaborate on the evaluation methods (cf. Sec. 5) we have used to gather the results presented in Section 6. The paper closes with a conclusion and ideas for future work.

2 Application

In this section we briefly introduce BibSonomy, the collaborative tagging system used to deploy our framework, define what a folksonomy is and how we can express some of its properties, and describe the tag recommendation task.

2.1 BibSonomy

As foundation and testbed for our framework we use the social bookmark and publication sharing system BibSonomy [Hotho et al., 2006a] which is run by us. BibSonomy started as a students project in spring 2005 and since then has evolved into a system with more than 1,500 active users. The goal was to implement a system for organizing BibTeX entries in a way similar to bookmarks in Delicious – which was at that time becoming more and more popular. After integrating bookmarks as a second type of resource into the system and upon the progress made, BibSonomy was opened for public access at the end of 2005 – first announced to colleagues only, later in 2006 to the public.

Users of BibSonomy can organize their bookmarks (URLs, favourites) and publication references by annotating them with tags. Plenty of features support them in their work: groups, tag editors, relations, various import and export options, etc. In particular, a REST-like [Fielding, 2000] API eases programmatic interaction with BibSonomy and is the cornerstone of external cooperation with the presented tag recommendation framework. Technically, BibSonomy is based on several Java modules which are merged in a Java Servlet/ServerPages based web application with an SQL database as backend.

http://www.bibsonomy.org/help/doc/api.html

Some of them are freely available at http://dev.bibsonomy.org/
regarded score from left to right. Thus, the recommender in action directly below the tag input box (in the screenshot the already entered letters. The suggested tags are shown all of the user’s previously used tags whose prefix matches reception which selects tags among the recommended tags (‘tags of copied item’). They are also regarded for autocompletion.

More formally, the tag recommendation task is: Given a resource \( r \) and a user \( u \) who wants to annotate \( r \), the recommender shall return a set of recommended tags \( T(u,r) := \{t_1, \ldots, t_k\} \) together with a scoring function \( f: T(u,r) \to [0,1] \) which assigns to each tag a score.\(^8\) The value of \( k \) is fixed to 5 throughout this paper.

3 Related Work

Although having a different recommendation target (resources rather than tags), the REFEREE framework described by Cosley et al. \cite{Cosley et al., 2006} is most closely related to our work. It provided recommendations for the CiteSeer (formerly ResearchIndex) digital library. REFEREE recommends scientific articles to users of ResearchIndex while they search and browse. An open architecture allows researchers to integrate their methods into REFEREE. Besides the different recommendation target, the focus of the work is more on the evaluation of several different strategies than on the details of the framework.

A powerful, open, and well documented framework for recommendations is the Duine Framework\(^9\) developed by Novay. It is based on work by van Setten et al. \cite{van Setten, 2002} and has a focus on explicit user ratings and non-occurring items, e.g., like in a movie recommendation scenario where one does not recommend movies the user has already seen. This is in contrast to tag recommendations, where re-occurring tags are a crucial requirement of the system. Similar to what we present in Section 4.2 the framework implements various hybrid recommenders. They have been studied extensively – for a survey see Burke \cite{Burke, 2002}.

Another recommendation framework is the AURA project’s ‘TasteKeeper’ \cite{Green and Alexander, 2006} from Sun Microsystems. Despite having not been described in the literature, it has a strong focus on collaborative filtering algorithms.

The topic of tag recommendations in social bookmarking systems has attracted quite a lot of attention in the last years. Most related work describes recommendation approaches which could be used within our framework. The existent approaches usually lay in the collaborative filtering and information retrieval areas \cite{Mishne, 2006, Byde et al., 2007, Sood et al., 2007, Xu et al., 2006} identify properties of good tag recommendations like high coverage of multiple facets, high popularity, or least-effort and introduce a collaborative tag suggestion approach. A goodness measure for tags, derived

\[^{8}\]Although, of course, \( f \) also depends on \( u \) and \( r \), we will omit these two variables to simplify notation. Since \( f \) always appears together with \( T(u,r) \), it should be clear from context, which \( f \) is meant.

\[^{9}\]In the original definition \cite{Hotho et al., 2006}, we introduced additionally a subtag/supertag relation, which we omit here.
from collective user authorities, is iteratively adjusted by a reward-penalty algorithm. Further examples include Basile et al. [Basile et al., 2007], suggesting an architecture of an intelligent tag recommender system, and Vojnović et al. [Vojnović et al., 2007], trying to imitate the learning of the true popularity ranking of tags for a given resource during the assignment of tags by users.

Heymann et al. [Heymann et al., 2008] model the tag prediction task as a binary classification problem for each tag with the web pages being the objects to classify. Besides the content of web pages, they also incorporate the anchor texts of links pointing to the page and host names of in-/out-links as features for a support vector machine (SVM). They try to answer questions like “What precision can we get with low recall?”, “Which page information is best for predicting tags?”, or “What makes a tag predictable?”. Additionally, they apply association rules between tags to expand tag-based queries. Another analysis of the application of classification methods to the tag recommendation problem can be found in [Ilig et al., 2009 to appear].

One task of the 2008 ECML PKDD Discovery Challenge [Hotho et al., 2008] also addressed the problem of tag recommendations in folksonomies. Tatu et al. [M. Tatu and D’Silva, 2008] base their suggestions on normalized tags from posts and normalized concepts from textual content of resources. This includes user added text like title or description as well as the document content. Using NLP tools they extract important concepts from the textual metadata and normalize them using Wordnet. Lipczak [Lipczak, 2008] developed a three step approach which utilizes words from the title expanded by a folksonomy driven lexicon, personalized by the tags of the posting user. Katakis et al. [I. Katakis and Vlahavas, 2008] consider the recommendation task as a multilabel text classification problem with tags as categories.

In [Jäschke et al., 2008] we evaluated several tag recommendation methods on three large scale folksonomy datasets. The most successful algorithm, the graph-based FolkRank [Hotho et al., 2006b], was followed by simpler approaches based on co-occurrence counts and by collaborative filtering.

4 A Recommendation Framework for BibSonomy

Implementing a tag recommendation framework requires to tackle several challenges. For example, having enough data available for recommendation algorithms to produce helpful recommendations is an important requirement. The recommender needs access to the systems database and to what the user is currently posting (which could be accomplished, e.g., by (re-)loading recommendations using techniques like AJAX). Further data – like the full text of documents – could be supplied to tackle the cold-start problem (e.g., for content-based recommenders). Further aspects which should be taken into account include implementation of logging of user events (e.g., clicking, key presses, etc.) to allow for efficient evaluation of the used recommendation methods in an online setting. Together with a live evaluation this also allows us to tune the result selection strategies to dynamically choose the (currently) best recommendation algorithm for the user or resource at hand. The multiplexing of several available algorithms together with the simple inclusion of external recommendation services (by providing an open recommendation interface) is one of the benefits of the proposed framework.

Figure 2 gives an overview on the components of BibSonomy involved in a recommendation process. The web application receives the user’s HTTP request and queries the multiplexer (cf. Sec. 4.4) for a recommendation – providing it post information like URL, title, user name, etc. Besides, click events are logged in a database (see Sec. 5.2). The multiplexer then requests the active recommenders to produce recommendations and selects one of the results. The suggested tags and the post are then logged in a database and the selected recommendation returned to the user.

4.1 Recommender Interface

One central element of the framework is the recommender interface. It specifies which data is passed from a recommendation request to one of the implemented recommenders and how they shall return their result. Figure 3 shows the UML class diagram of the TagRecommender interface one must implement to deliver recommendations to BibSonomy.

We decided to keep the interface as simple as possible by requiring only three methods, building on BibSonomy’s existing data model (Post, Tag, etc.) and adding as few classes as possible (RecommendedTag, RecommendTagComparator).

The getRecommendedTags method returns – given a post – a sorted set of tags; addRecommendedTags adds to a given (not necessarily empty) collection of tags further tags. Since – given a post and an empty collection – addRecommendedTags should return the same result as getRecommendedTags, the latter can be implemented by delegation to the former. Nonetheless, we decided to require both methods to cover the simple ‘give me some tags’ case as well as more sophisticated usage scenarios (think of ‘intelligent’ collection implementations, or a recommender which improves given recommendations).

The post given to both methods contains data like URL, title, description, date, user name, etc. that will later be stored in the database and that the recommender can use to produce good recommendations. It might also contain tags, i.e., when the user edits an existing post or when he has already entered some tags and requests new recommendations. Implementations could use those tags to suggest different tags or to improve their recommendation.

With the setFeedback method the final post as it is stored in the database is given to the recommender such that it can measure and potentially improve its performance. Additionally, the postID introduced in Section 5.3 is contained in the post (as well as in the post of the first two methods) such that the recommender can connect the post with the recommended tags it provided.

Finally, the getInfo method allows the programmer to
provide some information describing the recommender. This can be used to better identify recommenders or be shown to the user.

Two further classes augment the interface: The `RecommendedTag` class basically extends the `Tag` class as used in the BibSonomy API (cf. Sec. 3) by adding floating point `score` and `confidence` attributes. A corresponding `RecommendedTagComparator` can be used to compare tags, e.g., for sorted sets. It first checks textual equality of tags (ignoring case) and then sorts them by score and confidence. Consequently, tags with equal names are regarded as equal.

Our implementation is based on Java and all described classes are contained in the module `bibsonomy-model`, which is available online as JAR file in a Maven2 repository.\footnote{\url{http://dev.bibsonomy.org/maven2/org/bibsonomy/bibsonomy-model/}} However, implementations are not restricted to Java – using the remote recommender (see Sec. 4.3) one can implement a recommender in any language which is then integrated using XML over HTTP requests.

### 4.2 Meta Recommender

Meta or hybrid recommenders \cite{Burke2002} do not generate recommendations on their own but instead call other recommenders and modify or merge their results. Since they implement the same interface, they can be used like any other recommender. More formally, given \( n \) recommendations \( T_1(u, r), \ldots, T_n(u, r) \) and corresponding scoring functions \( f_1, \ldots, f_n \), a meta recommender produces a merged recommendation \( T(u, r) \) with scoring function \( f \). The underlying design pattern known from software architecture is that of a Composite.

As we will see in Section 4.5, meta recommenders allow the building of complex recommenders from simpler ones and thus simplify implementation and testing of algorithms and even stimulate development of new methods. Furthermore, they allow for flexible configuration, since their underlying recommenders can be exchanged at runtime. This section introduces the meta recommenders that are currently used in our framework.

#### First Weighted By Second

As an example of a cascade hybrid, the idea behind this recommender is to re-order the tags of one recommendation using scores from another recommendation. More precisely, given recommendations \( T_1(u, r) \) and \( T_2(u, r) \) and corresponding scoring functions \( f_1 \) and \( f_2 \), this recommender returns a recommendation \( T(u, r) \) with scoring function \( f \), which contains all tags from \( T_1 \) which appear in \( T_2 \) (with \( f(t) := f_2(t) \)) plus all the remaining tags from \( T_1 \) (with lower \( f \) but respecting the order induced by \( f_1 \)). If \( T_1(u, r) \) does not contain enough recommendations, \( T \) is filled by the not yet used tags from \( T_2(u, r) \) – again with \( f \) being lower than for the already contained tags and respecting the order induced by \( f_2 \).

#### Weighted Merging

This weighted hybrid recommender enables merging of recommendations from different sources and weighting of their scores. Given \( n \) recommendations \( T_1(u, r), \ldots, T_n(u, r) \), corresponding scoring functions \( f_1, \ldots, f_n \), and (typically fixed) weights \( \rho_1, \ldots, \rho_n \) (with \( \sum_{i=1}^{n} \rho_i = 1 \)), the weighted merging recommender returns a recommendation \( T(u, r) := \bigcup_{i=1}^{n} T_i(u, r) \) and a scoring function \( f(t) := \sum_{i=1}^{n} \rho_i f_i(t) \) (with \( f_i(t) := 0 \) for \( t \notin T_i(u, r) \)).

### 4.3 Remote Recommender

The remote recommender retrieves recommendations from an arbitrary external service using HTTP requests in REST-based \cite{Fielding2000} interaction. Therefore, it uses the XML schema of the BibSonomy REST-API.\footnote{\url{http://dev.bibsonomy.org/maven2/org/bibsonomy/bibsonomy-recommender-servlet}} This recommender has three advantages: it allows us to distribute the recommendation work over several machines, it opens the framework to include recommenders from auxiliary partners, and it enables programming language independent interaction with the framework.

To simplify implementation of external recommenders, we provide an example web application needing almost zero configuration to include a custom Java recommender.\footnote{\url{http://dev.bibsonomy.org/maven2/org/bibsonomy/bibsonomy-recommender-servlet}} Furthermore, we plan to integrate recommendations into BibSonomy’s API to allow clients retrieve recommendations (e.g., such that the Firefox browser add-on can show recommendations during bookmark posting).

### 4.4 Multiplexing Tag Recommender

Our framework’s technical core component is the so-called multiplexing tag recommender (see Fig. 2). Implementing BibSonomy’s tag recommender interface, it provides the web application with tag recommendations, using one of the recommenders available. All recommendation requests and each recommender’s corresponding result are logged in a database (see Sec. 5.3). For this purpose, every tag recommender is registered during startup and assigned to a unique identifier. For technical reasons, we differentiate between locally installed and remote recommenders (cf. Sec. 4.3).

Whenever the `getRecommendedTags` method is invoked, the corresponding recommendation request is delegated to each recommender, spawning separate threads for each recommender. After a timeout period of 100 ms, one of the collected recommendations is selected, applying a pre-configured selection strategy:

For our evaluation process we implemented a ‘sampling with replacement’ strategy, choosing exactly one recommender \( i \) and all of its recommended tags \( T_i(u, r) \) together...
with its scoring function $f_1$. If the user requests tag recommendations more than once (e.g., using the ‘reload’ button), this process is repeated independently from previous requests.

### 4.5 Example Recommender Implementations

Using the proposed framework, we implemented several recommendation methods, whereas two of them are currently active in BibSonomy. Both build upon the meta recommenders described in Section 4.2 and simpler recommenders which we describe only briefly because they are fairly self-explanatory. The short names in parentheses are for later reference.

**Most Popular $\rho$-Mix (MP$\rho$-mix)**

Motivated by the good results of mixing tags which often have been attached to the resource with tags the user has often used, we implemented a variant of the most popular $\rho$-mix recommender described in [Jäschke et al., 2008]. The recommender has been implemented as a combination of three recommenders, using a value of $\rho = 0.6$:

1. the most popular tags by resource recommender which returns the $k$ tags $T_1(u, r)$ which have been attached to the resource most often (with $f_1(t) := \frac{|Y \cap T \times \{r\}|}{|Y \times \{r\}|}$, i.e., the relative tag frequency),
2. the most popular tags by user recommender which returns the $k$ tags $T_2(u, r)$ the user has used most often (with $f_2(t) := \frac{|Y \cap (\{u\} \times T)|}{|\{u\} \times T|}$, i.e., the relative tag frequency), and
3. the weighted merging meta recommender described in Section 4.2 which merges the tags of the two former recommenders, with weights $\rho_1 = \rho = 0.6$ and $\rho_2 = 1 - \rho = 0.4$.

**Title Tags Weighted by User Tags (TbyU)**

Inspired by the first recommender implemented in BibSonomy [Illig, 2006] and by similar ideas in [Lipczak, 2008], we implemented a recommender which ranks tags extracted from the resource’s title using the frequency of the tags used by the user. Technically, this is again a combination of three recommenders:

1. a simple content based recommender, which extracts $k$ tags $T_1(u, r)$ from the title of a resource, cleans them and checks against a multilingual stopword list,
2. the most popular tags by user recommender as described in the previous section – here returning all tags $T_2(u, r)$ the user has used (by setting $k = \infty$), and
3. the first weighted by second meta recommender described in Section 4.2 which weights the tags from the content based recommender by the frequency of their usage by the user as given by the second recommender.

**Other**

Besides the simple recommenders introduced along the MP$\rho$-mix and TbyU recommender, we have implemented recommenders for testing purposes (a fixed tags recommender and a random tags recommender), a recommender which proposes tags from a web page’s HTML meta information keywords, as well as a recommender using the FolkRank algorithm [Hotho et al., 2006b].

More complex recommenders can be thought of, e.g., a nested first weighted by second recommender, whose first recommender is a weighted merging meta recommender merging the suggestions from a content based recommender and a most popular tags by resource recommender and then scoring the tags by the scores from the most popular tags by user recommender.

### 5 Evaluation

We evaluate the performance of a recommender by comparing the tags it suggested with the tags used to annotate a resource. Then recall (‘Which fraction of the used tags could be suggested?’) and precision (‘Which fraction of the suggested tags was used?’) quantify the quality of the recommendation. Furthermore, the logging of click events allows us to evaluate the user behavior in more detail.

#### 5.1 Measures

As performance measures we use precision, recall, and f1-measure ($f1m$) which are standard in such scenarios [Herlocker et al., 2004]. For each post $(u, T_{ur}, r)$ we compare the recommended tags $T(u, r)$ with the tags $T_{ur}$ the user has finally assigned. Then, precision and recall of a recommendation are defined as follows

\[
\text{precision}(T(u, r)) = \frac{|T_{ur} \cap T(u, r)|}{|T_{ur}|} \quad (1)
\]
\[
\text{recall}(T(u, r)) = \frac{|T_{ur} \cap T(u, r)|}{|T(u, r)|}. \quad (2)
\]

We then average these values over all posts in the given set and compute the f1-measure as

\[
f1m = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} .
\]

#### 5.2 Data Cleansing

Before intersecting $T_{ur}$ with $T(u, r)$, we clean the tags in both sets according to the Java method cleanTag shown in Listing 1. This means, we ignore the case of tags and remove all characters which are neither numbers nor letters.

Since we assume all characters to be UTF-8 encoded, the method will not remove umlauts and other non-latin characters. We also employ unicode normalization to normal form KC. Finally, we ignore tags which are ‘empty’ after normalization (i.e., they neither contained a letter nor number) or which are equal to the strings imported, public, systemimported, nn, systemunfiled. Thus, in the following we always regard cleaned tags.

---

**Listing 1:** The Java method used to clean tags.

```java
public String cleanTag(String tag) {
    return Normalizer.normalize(tag, Normalizer.Form.NFKC).
        replaceAll("[^0-9\p{L}]", "").
    toLowerCase();
}
```
5.3 Logging

For evaluating performance of the tag recommenders available, we store in a database for each recommendation process the corresponding bookmark or BibTeX entry as well as each recommender’s recommendation, identified by a unique recommendationID. Furthermore, the applied selection strategy together with the recommenders and tags selected are stored.

Several recommendation requests may refer to a single posting process (e.g., when the user pressed the ‘reload’ button). For identifying these correspondences, a random identifier (postID) is generated whenever a post or editing process is started and retained valid until the corresponding post is finally stored in BibSonomy. This postID is mapped to each corresponding recommendationID. At storage time, the postID together with the corresponding user name, time stamp and a hash identifying the resource is stored. This connects each post of each user with all referring recommendations and vice versa.

Additionally, the user interaction is tracked by logging mouse click events using JavaScript. Each click on one of BibSonomy’s web pages is logged using AJAX into a separate logging table. Information like the shown page, the DOM path of the clicked element, the underlying text, etc is stored.

6 Results

The following analysis is based on data from posting processes between April 8th and May 8th 2009; this is ongoing work – this analysis is the first step of a long term study of the BibSonomy recommendation framework. Only public posts from users not flagged as spammer were taken into account. Since tag recommendations are provided in the web application only when one resource is posted, posts originating from automatic import (e.g., Firefox bookmarks, or BibTeX files) or BibSonomy’s API are not contained in the analysis.

6.1 General

We start with some general numbers: In the analysed period, 5,840 posting processes (3,474 for BibTeX, 2,366 for bookmarks) have been provided with tag recommendations. The MP\textsubscript{p}-mix recommender served recommendations for 2,935 postings, the TbyU recommender for 3,006.

For each recommender \(r\), we can then look at the sets \(F_r \setminus L_r\), \(L_r \setminus F_r\), and \(F_r \cap L_r\). Posits where the user did not press the reload button are contained in both \(F_r\) and \(L_r\), and thus in \(F_r \cap L_r\). Table 1 shows the result of our analysis.

### Table 1: The influence of the ‘reload’ button.

<table>
<thead>
<tr>
<th>recommender (r)</th>
<th>(MP_p\textsubscript{mix})</th>
<th>TbyU</th>
<th>(MP_p\textsubscript{mix})</th>
<th>TbyU</th>
</tr>
</thead>
<tbody>
<tr>
<td>(F_r \setminus L_r)</td>
<td>337</td>
<td>319</td>
<td>0.258</td>
<td>0.270</td>
</tr>
<tr>
<td>(L_r \setminus F_r)</td>
<td>331</td>
<td>363</td>
<td>0.380</td>
<td>0.364</td>
</tr>
<tr>
<td>(F_r \cap L_r)</td>
<td>2,271</td>
<td>2,339</td>
<td>0.277</td>
<td>0.224</td>
</tr>
</tbody>
</table>

Several recommenders can be involved in one posting process. There is the recommendation which appears directly after loading the posting page (first), there are recommendations which appear after the user has pressed the ‘reload’ button, and there is the recommendation shown before the user finally saves the post (last). Thus, given a recommender \(r\), we can define the set \(F_r\) to contain those posts, where the recommender \(r\) showed the first tags, and \(L_r\) as the set of posts where recommender \(r\) showed the last tags (i.e., before the post is stored).

For each recommender \(r\) we can then look at the sets \(F_r \setminus L_r\), \(L_r \setminus F_r\), and \(F_r \cap L_r\). Posits where the user did not press the reload button are contained in both \(F_r\) and \(L_r\), and thus in \(F_r \cap L_r\). Table 1 shows the result of our analysis.

For both of the two deployed recommenders and for all three sets, the table shows the number of posts in the corresponding set, and the average f1-measure at the fifth tag. As one can see, the number of posts where the reload button has not been pressed \((F_r \cap L_r)\) is quite large for both recommenders (almost 1800). There is also only little difference in the number of posts for the recommenders over the different sets, besides the clearly higher number of posts for the \(MP_p\textsubscript{mix}\) recommender in the \(F_r \setminus L_r\) set. It contains those posts where the user requested to reload the recommendations and where the recommender at hand delivered the first recommendation. Thus, the \(MP_p\textsubscript{mix}\) recommender more often caused the user to press the reload button than on the TbyU recommender’s suggestion. The slightly higher f1-measure of the TbyU recommender for this set furthermore suggests that its recommendations were even better in this case.

The most noticeable observation is the good performance of the \(MP_p\textsubscript{mix}\) recommender for the posts where a reload occurred and this recommender showed the last recommendation.

### Figure 4: Precision and Recall

![Figure 4: Precision and Recall](image)

Note that users can disable logging on the settings page, thus not all posting processes yield clicklog events.

---

**Note:** We omit precision and recall, since whenever f1@5 for one set was better/worse than for another set, precision and recall were better/worse, too.
ommendations ($L_T \setminus F_i$). Here both precision and recall are much better than for the other two sets. A similar but not so strong observation can be made for the TbyU recommender for the same set. However, this set is also the only one which shows a significant difference between the two recommenders. The better performance of the MP$r$-mix recommender could be an indicator that it provided the better recommendations. In any case, the recommendations shown last ($L_T \setminus F_i$) are better than those shown first ($F_i \setminus L_T$).

The usage of the ‘reload’ button is a good indicator for the interest of the user in the recommendations. When more recommenders are deployed together with a ‘sampling without replacement’ strategy (cf. Sec. 4.2), there will probably be a more significant variation among the recommenders and sets.

6.3 Logged ‘click’ Events

Next we evaluate the data from the log which records when a user clicked on a recommended tag (cf. Sec 5.3). Clicks are rather sparse: in only 1,061 (485 bookmark, 576 BibTeX) of the 5,840 posting processes users clicked on a recommendation.

In Figure 5 we see for each number of recommended tags (from one to five), the fraction of matches which stem from a click on the tag (instead of manual typing). For the TbyU recommender almost 35 % of the matches when regarding only the first tag come from clicking on it. Thus, although users infrequently click on tags, a large fraction of the correctly recommended tags has been clicked instead of typed.

Next we want to answer the questions “How is clicking distributed over users?”, “Are there users which always/n- ever click?”. Figure 6 shows users sorted by the fraction of posting processes at which they have clicked on a recommended tag. The size of each circle depicts the logarithm of the user’s number of posts. Closer to the left are users which in almost all posting events clicked on a recommendation; users closer to the right almost never clicked a tag during recommendation. Although only around 150 users clicked on a recommendation, half of the remaining users are represented by only one post. This could mean that only after some time users discover and use the recommendations. However, there are also some active users which almost never clicked on a recommendation.

6.4 Average F1-Measure per User

Which properties of a posting process could help a multiplexer strategy to smartly choose a certain recommender instead of randomly selecting one? For space reasons we

Figure 5: The fraction of matching tags which have been clicked.

Figure 6: Users sorted by their fraction of click/no-click posts; The y-axis depicts the fraction of posts where recommended tags were clicked.

Figure 7: Average f1-measure for each user and recommender.

focus on the user only – other characteristics could be likewise interesting (e.g., resource type or the recommended tags). Figure 7 shows the average f1m of the MP$r$-mix recommender versus the average f1m of the TbyU recommender for each of the 284 users in the data. In the plot, each user is represented by a circle whose size depicts the logarithm of the user’s number of posts.

The most interesting users are reflected by the circles farthest from the diagonal, i.e., those users who have a high f1m for one but a low f1m for the other recommender. As one can see, such users exist even at higher post counts. Once such a user is identified, one could primarily select recommendations from the user’s preferred recommender.

7 Conclusions and Future Work

In this paper, we presented the tag recommendation framework we developed for BibSonomy. It allows us to not only integrate and judge recommendations from various sources but also to develop clever selection strategies. A strength of the framework is its ability to log all steps of the recommendation process and thereby making it traceable. E.g., the diagrams and tables presented in this paper are automatically generated and will be integrated in a web application for analysing and controlling the framework and its recommenders.

As the results show, there is no clear picture which of the two recommendation methods performs better. There is a dependency on the number of regarded tags, the user

13 Only users which got recommendations from both recommenders were taken into account.
at hand, and also slightly on the moment of recommendation. This suggests that we can achieve better performance not only by adding improved recommendation methods but also by implementing adaptive selection strategies. In case of the user dependency, one could prefer the better performing recommender by increasing its selection probability or even couple the probability with the current recommendation quality.

Finally, the framework will be the cornerstone of this year’s ECML PKDD Discovery Challenge[18] where one task requires the participants to deliver live recommendations for BibSonomy. This will also be a larger stress test for external recommenders and the framework itself. After that we will open the framework for interested parties.

Acknowledgement Part of this research was funded by the European Union in the Tagora (FET-IST-034721) project and by the DFG in the project “Info 2.0 – Informationelle Selbstbestimmung im Web 2.0”.

References


