FReSET - An Evaluation Framework for Folksonomy-based Recommender Systems

Renato Domínguez García
renato@kom.tu-darmstadt.de

Matthias Bender
mbender@kom.tu-darmstadt.de

Mojisola Anjorin
anjorin@kom.tu-darmstadt.de

Christoph Rensing
rensing@kom.tu-darmstadt.de

Ralf Steinmetz
ralf.steinmetz@kom.tu-darmstadt.de

Multimedia Communications Lab
TU Darmstadt
64283 Darmstadt, Germany

ABSTRACT

FReSET is a new recommender systems evaluation framework aiming to support research on folksonomy-based recommender systems. It provides interfaces for the implementation of folksonomy-based recommender systems and supports the consistent and reproducible offline evaluations on historical data. Unlike other recommender systems framework projects, the emphasis here is on providing a flexible framework allowing users to implement their own folksonomy-based recommender algorithms and pre-processing filtering methods rather than just providing a collection of collaborative filtering implementations. FReSET includes a graphical interface for result visualization and different cross-validation implementations to complement the basic functionality.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms

Algorithms, Experimentation, Measurement

1. INTRODUCTION

The amount of documents and information on the Internet is growing steadily. In particular through Web 2.0 applications more and more information is made available to users. A lot of this information can be interesting or helpful to users, but due to information overload it can be difficult for them to find relevant items. Recommender systems can be used for automatic filtering of relevant information. Therefore the research on recommender systems is getting more and more important for Web 2.0 communities. A lot of these communities are social resource sharing networks: They can be used to share bookmarks like delicious[1], to share photos like flickr[2] or to share music listening habits like Last.fm[3]. Even if the type of the shared resources are different, all these networks are very similar: users can add resources to the network and tag the resources with arbitrary words. In this way so-called *Folksonomies* emerge. Folksonomies are defined as the collection of users, tags and resources in a community. In the last years, a lot of research in folksonomy-based recommender systems has been done [4, 6]. However, if we take a look on existing recommender system frameworks like Cofi[4], LensKit[2] or recommenderlab[5], we see that these frameworks focus on the development and applications of collaborative filtering techniques, which are very simple and effective. However, these frameworks cannot be applied directly to support research on folksonomy-based recommender systems. Folksonomies differ in essence from traditional recommender data models by the use of tags and the non-existence of ratings, i.e. whereas traditional recommender systems usually operate over 2-dimensional data arrays, folksonomies are represented as graphs with edges denoting (user, resource, tags) triples. Furthermore, in folksonomy-based recommender systems we are interested not only in content, but also in tags and even other users. Additionally, tagging can be extended to semantic tagging, where tags are enriched by semantics, e.g. describing their type [1].

FReSET[6] aims to close this gap by providing interfaces and methods to implement, analyze and compare recommender systems. FReSET also offers a graphical interface in order to get a better visualization of the evaluation data. In the next sections we will describe the functionality to pre-process, evaluate and visualize the data in the evaluation framework.

2. THE FRAMEWORK

[1]http://www.delicious.com (accessed 08.06.2012)
[2]http://www.flickr.com (accessed 08.06.2012)
[3]http://www.lastfm.com (accessed 08.06.2012)
[4]http://www.nongnu.org/cofi (accessed 08.06.2012)
[5]http://lyle.smu.edu/IDA/recommenderlab (accessed 08.06.2012)
[6]Folksonomy-based REcommender System Evaluation Tool

2.1 Architecture

Our framework consists of four main packages that encapsulate different parts of its functionality. These packages shall be described in detail in the following sections.

2.1.1 *Folksonomy Data Model and File Format*

In our framework, folksonomies are designed as a graph consisting of vertices and edges connecting them. The package `folksonomy` contains all classes needed. The central functionality of the data structure can be found in the abstract class `Vertex`. It simply has a set of edges (represented by the class `Edge`) and through them knows about its neighbors. The classes `Resource`, `Tag`, and `User` extend it. In order to use them effectively there is the class `Folksonomy`. It not only gives access to all vertices and edges but manages adding, connecting and removing them.

FReSET input files follow a simple plain-text format. Each line in the file represents one post. A post consists of a resource r, the user u who added the resource and all tags t that the user has assigned to r. The elements (resource, tag(s), user) are delimited using TAB. The first string represents the URL of a resource (e.g. `bibsonomy.org`), the second the tag(s) separated by comma (e.g. `bookmarking,social`) and the third the user who has tagged the resource (e.g. `matthiasb`). Optionally, a fourth string representing the date the resource was added can be specified too. In Table 1 is an example of a folksonomy represented in our file format:

bibsonomy.org	bookmarking,social	matthiasb
flick.com	action:tagging, media:social	renato
youtube.com	media:videos,tagging,social	moji

Table 1: A very simple FReSET input file

Furthermore, folksonomy graphs can be extended using edges between any two vertices, for example to represent friendship relationships between users, similarity measures between resources or semantic relations between tags. The input format for these relations consists of a capital letter representing the vertex type, i.e. resource (R), tag (T) and user (U) followed by its name. An example for additional edges is shown in Table 2:

Tfotos	Tvideos
Tsocial	Rbibsonomy.org
Umatthiasb	Urenato

Table 2: A very simple FReSET input file for additional edges

2.1.2 *Recommendation Tasks*

The framework supports three types of recommendation tasks which are given as interfaces in the package `recommender`:

1. The tag recommendation task, which finds the best fitting tags for a given user. It is represented by the interface `TagRecommender`.

2. The resource recommendation task, which calculates a list of resources for a given user described by the interface `ResourceRecommender`.

3. The user recommendation task that returns users similar to a given user. The interface `UserRecommender` defines this recommendation task.

Each interface inherits the `Recommender` interface which defines a recommender by one method. This method gets a folksonomy and a user and will return a list of vertices. In order to evaluate a recommender, this method has to be implemented as one of the three previously described recommendation tasks. Figure 1 gives an overview of all interfaces.

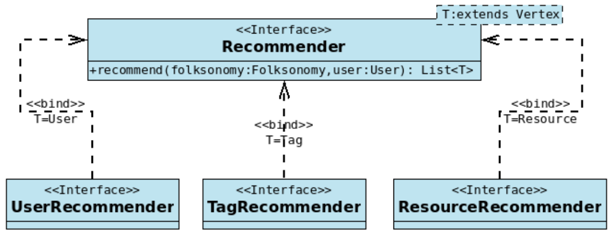


Figure 1: Modular structure of recommenders

2.1.3 *Evaluation*

All functionality needed in order to evaluate recommenders can be found in the package `evaluation`. The framework focuses its evaluation techniques on an abstract concept we call evaluators. An evaluator is defined by the abstract class `Evaluator`. This class provides the basic structure to implement different types of evaluations on a single recommender.

In general, recommender systems are evaluated using evaluation metrics like precision, recall or mean absolute error, which are implemented in FReSET. Thus the primary output of an evaluator has to be numeric. However for plotting graphs or integrating these values into some sort of summary text, the results have to be more complex. For this reason, the method `evaluate` which gets a recommender and a folksonomy will return an instance of `EvaluationResult`. In its most basic form it simply provides a summary string and a list of graphical representations.

The technique used most often to evaluate recommender systems is Cross-validation [5]. Since a folksonomy cannot easily be split into n folds of equal sizes, a variant of a Leave-One-Out Cross-validation is applied. Thus, the most important extension of `Evaluator` may be the class `CrossValidation`. It is the generic and abstract implementation of a per-user Cross-validation. Most of the types of Cross-validation on folksonomies can be reduced to this basic algorithm. The algorithm calculates for every user in the folksonomy, a list of folds. A fold consists of two parts: the list of edges and vertices removed from the folksonomy and the list of vertices to validate the recommendation against. For any of these folds, a single evaluation of a recommender is now applied. Currently the following variations of Cross-validation are already implemented:

Leave-Post-Out: This method removes in each fold for a user one of his resources as well as the concurrent edges of the resource

Leave-Tag-Out: Analogue to the Leave-Post-Out Cross-validation the Leave-Tag-Out Cross-validation can be applied to resource recommenders. Instead of iterating over all resources, it leaves each tag connected to a user out and validates against the resources connecting the user and tag

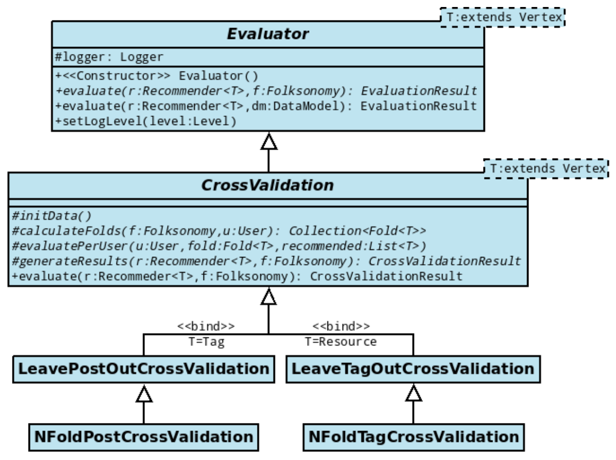

T:extends Vertex
Evaluator
#logger: Logger
+<<Constructor>> Evaluator()
+evaluate(r:Recommender<T>,f:Folksonomy): EvaluationResult
+evaluate(r:Recommender<T>,dm:DataModel): EvaluationResult
+setLogLevel(level:Level)
T:extends Vertex
CrossValidation
#initData()
#calculateFolds(f:Folksonomy,u:User): Collection<Fold<T>>
#evaluatePerUser(u:User,fold:Fold<T>,recommended:List<T>)
#generateResults(r:Recommender<T>,f:Folksonomy): CrossValidationResult
+evaluate(r:Recommeder<T>,f:Folksonomy): CrossValidationResult
<<bind>>
T=Tag
<<bind>>
T=Resource
LeavePostOutCrossValidation
LeaveTagOutCrossValidation
NFoldPostCrossValidation
NFoldTagCrossValidation

Figure 2: Examples of classes inheriting `Evaluator`

N-Folds-per-User: Another approach ensures there are a maximum of N folds used for each user. Again there are two variations: one using posts for tag recommenders `NFoldPostCrossValidation` and another that leaves tags out for resource recommenders `NFoldTagCrossValidation`.

2.2 Features

In the following section we elaborate some of the other functions the framework offers. These parts mostly exist in order to make the framework more accessible and create an environment in which experiments can be performed.

2.2.1 *Pre-processing and Parameterization*

Typos in vertex names often lead to vertices having very few connections to other vertices in the folksonomy. This influences the evaluation of recommenders since there is no way those vertices could be predicted. These and other influences often make it necessary to pre-process the data before using it to evaluate a recommender. Inspired by the concept of filters in Weka [3], we introduce a similar concept in our framework. The interface `Filter` can be found in the package `preprocess`. It declares a method `filterFolksonomy` which gets a folksonomy and returns the modified object. As simple as that, it can basically perform every possible operation on folksonomies.

The only restriction to this concept is the lack of dynamics. Whenever a filter needs more information than the folksonomy alone can provide, this concept will no longer work. In order to fix this, we introduce the package `parameterization`. It adds a class `Parameter` and an immutable list `ParameterList` that also serves as a type of map by enabling the direct search for a parameter given its name. A parameter has a type, a name, and a default value. Right now there are only four types available: String, Integer, Float and Boolean. Additionally, an event based system exists that can be used to ensure a parameter fulfills certain constraints. Even though it is not primarily needed there, evaluators and recommenders also benefit from the parametrization concept. Figure 3 visualizes these relationships.

2.2.2 *Graphical User Interface*

Even though the framework can directly be used as a library in Java or to implement own experiments, there is

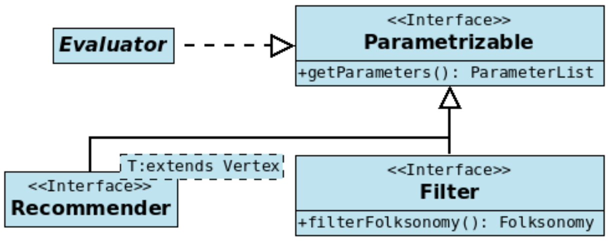
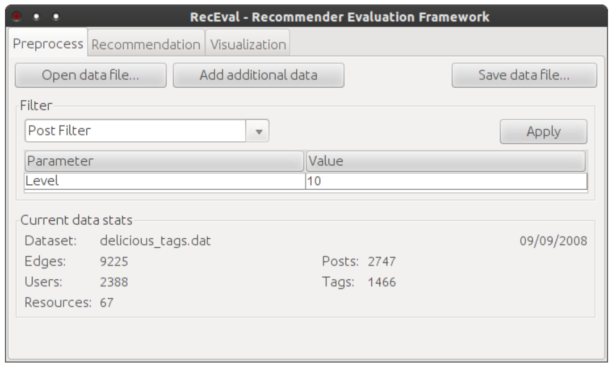


Figure 3: The parameterizable interface and where it's applied

an additional package containing a graphical user interface. It provides an environment that leads step by step through the evaluation process. This user interface consists of three parts which shall now be discussed in detail:

The first part when starting the program is the pre-processing panel shown in Figure 4. It gives access to the functionality needed to load folksonomies into the program, to modify them and to write them back to a file. After opening a file, some statistics about the current folksonomy will be shown at the bottom of the tab. The middle part allows modifications through the filters described earlier. Having



Filter

Post Filter

Parameter	Value
Level	10

Current data stats

Dataset: delicious_tags.dat 09/09/2008
Edges: 9225 Posts: 2747
Users: 2388 Tags: 1466
Resources: 67

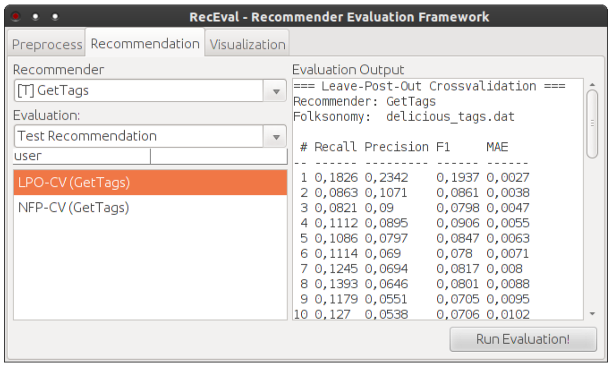
Figure 4: The pre-processing panel

prepared a folksonomy, the recommender selection panel can now be opened. This panel allows to perform the evaluation experiments and to display textual results. At first the recommender has to be selected and configured. After that the evaluation method can be selected. The panel will log the texts and and all results are displayed in a list. This is shown in Figure 5. Whenever an evaluation method produces numerical results, these are displayed on the last panel (see Figure 6). Its sole purpose is the visualization of the evaluation results. The graphs will also be grouped according to what they display.

2.2.3 *Example*

Listing 1 gives an example of how to use the FReSET framework for evaluating an implemented recommender on a given folksonomy. The first step is to instantiate the recommender. In our case we create an instance of a simple tag-recommender. Since every evaluation has to be based upon a dataset, the second step is to load a folksonomy from a file that has the format described earlier.

```
// create a new instance of the recommender
TagRecommender r = new GetTags();
```



Recommender: [T] GetTags
Evaluation: Test Recommendation
user
LPO-CV (GetTags)
NFP-CV (GetTags)

Evaluation Output

```
=== Leave-Post-Out Crossvalidation ===
Recommender: GetTags
Folksonomy:  delicious_tags.dat

 # Recall Precision F1     MAE
-- ------ --------- ------ ------
 1 0,1826 0,2342    0,1937 0,0027
 2 0,0863 0,1071    0,0861 0,0038
 3 0,0821 0,09      0,0798 0,0047
 4 0,1112 0,0895    0,0906 0,0055
 5 0,1086 0,0797    0,0847 0,0063
 6 0,1114 0,069     0,078  0,0071
 7 0,1245 0,0694    0,0817 0,008
 8 0,1393 0,0646    0,0801 0,0088
 9 0,1179 0,0551    0,0705 0,0095
10 0,127  0,0538    0,0706 0,0102
```

Figure 5: The recommender selection panel

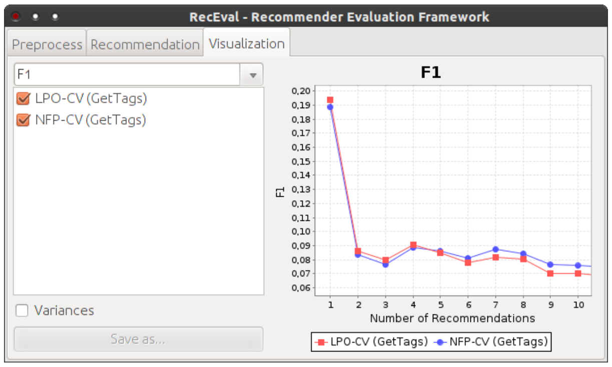


Figure 6: The result visualization panel

```
// load folksonomy from file
FolksonomyParser parser = new SplittParser();
Folksonomy f = parser.parseFolksonomyFile(
        new File("someFile.dat"));

// instantiate an evaluator
Evaluator<Tag> eval =
        new LeavePostOutCrossValidation();

// run the evaluation
EvaluationResult result = eval.evaluate(r, f);

// print summary string.
System.out.println(result.getSummaryString());
```

Listing 1: An example evaluation that prints its results to the terminal

At this point we need to choose and instantiate an evaluation method. This evaluation method has to fit the type of recommender and thus has to be an evaluator working on tag-recommenders. Finally, the evaluate method can be called which will return a result object. In the given example, only the resulting object is used to print out a textual summary of the results. But obviously it is also possible to directly access the resulting values.

2.2.4 *Demostration and Download*

The demostration will show the different features of FReSET by implementing on-the-fly a simple filter and a simple recommender. After that the simple recomemender will be evaluated and compared with a standard collaborative filtering recommeder system. If you are interested in obtaining the FReSET framework, please send an e-mail to renato@kom.tu-darmstadt.de. From August, it is planned to offer a download link at `http://www.kom.tu-darmstadt.de/en/research-results/downloads/software/`.

3. CONCLUSION AND FUTURE WORK

In this paper, we described the FReSET framework which supports the evaluation and comparison of folksonomy-based recommender systems. The framework allows the implementation of recommenders and filters for pre-processing data using given interfaces. It provides additionally a GUI for the visualization and comparison of evaluation results. In the future we will implement additional standard algorithms like variants of collaborative filtering implementations, graph-based approaches like FolkRank and pre-processing filters. Further, we plan to add more possibilities for the parametrization of recommender systems, for example to not only recommend resources, users or tags taking a single user as input, but also taking resources or tags or combinations of these as multi-input recommendation tasks. Finally, we will introduce more visualization possibilities like ROC curves and significance tests.

4. REFERENCES

[1] D. Böhnstedt, L. Lehmann, C. Rensing, and R. Steinmetz. Automatic identification of tag types in a resource-based learning scenario. In *Towards Ubiquitous Learning*, volume 6964 of *LNCS*, pages 57–70. Springer Berlin, 2011.

[2] M. D. Ekstrand, M. Ludwig, J. Kolb, and J. Riedl. Lenskit: a modular recommender framework. In B. Mobasher, R. D. Burke, D. Jannach, and G. Adomavicius, editors, *RecSys*. ACM, 2011.

[3] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009.

[4] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in folksonomies. In J. Kok, J. Koronacki, R. Lopez de Mantaras, S. Matwin, D. Mladenic, and A. Skowron, editors, *Knowledge Discovery in Databases: PKDD 2007*, volume 4702 of *LNCS*, pages 506–514. Springer, 2007.

[5] L. Liu and M. T. Zsu. *Encyclopedia of Database Systems*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[6] L. B. Marinho, A. Hotho, R. Jäschke, A. Nanopoulos, S. Rendle, L. Schmidt-Thieme, G. Stumme, and P. Symeonidis. Social tagging systems. In *Recommender Systems for Social Tagging Systems*, Springer, pages 3–15. Springer US, 2012.