Towards Fully Distributed and Privacy-preserving Recommendations via Expert Collaborative Filtering and RESTful Linked Data

Jae-wook Ahn, University of Pittsburgh, Pittsburgh, USA
Xavier Amatriain, Telefonica Research, Barcelona, Spain

***Abstract*—Expert Collaborative Filtering is an approach to recommender systems in which recommendations for users are derived from ratings coming from domain experts rather than peers. In this paper we present an implementation of this approach in the music domain. We show the applicability of the model in this setting, and show how it addresses many of the shortcomings in traditional Collaborative Filtering such as possible privacy concerns. We also describe a number of technologies and an architectural solution based on REST and the use of Linked Data that can be used to implement a completely distributed and privacy-preserving recommender system.**

I. Introduction

Recommender systems can be seen as a practical alternative to traditional search. They can satisfy the user need for relevant information without the overhead of having the user explicitly state a query. The query is therefore derived from both the user preferences and the application context. Recommender systems have proved their business value in many contexts already, ranging from traditional e-Shopping sites [13] to very different settings such as Television [17].

One of the most favored approaches to recommending is Collaborative Filtering (CF). CF is a technique to filter or evaluate items through the opinions of other people [22]. It makes use of peer user ratings in order to provide recommendations on the items that are unknown but may interest the target user. Both the CF approach and a number of its shortcomings are explained in Sec. II

In our previous work [2], we introduced a radically different approach to CF in which recommendations are drawn from a pool of domain *experts*, instead of using the opinion of the general population. This *expert collaborative filtering* is explained in Sec. III. We also highlight how it addresses some of the main limitations in collaborative filtering.

Since the original study was based on the specific domain of movie recommendations, there were also some concerns that expert CF would not be applicable in other domains. For this reason, we developed our current application to implement the approach in the music domain. To understand the properties of this particular domain, we analyze the data obtained from music expert ratings in Sec.IV.

The original design of our previous experimental evaluation did not include the development of an application, so there were many unanswered questions related to the practicality of the approach. To address these issues, we now present a distributed application that uses latest web technologies such as REST [10] and Linked Data [7] to offer a practical implementation of expert CF. We describe the architecture of the application in Sec. V.

Therefore, the main contributions of this paper can be summarized in the following:

- A demonstration of the validity of expert CF in domains other than cinema
- An analysis of the different rating behavior between online movie and music critics
- A practical implementation of a completely distributed and privacy-preserving recommender system using latest web technologies such as REST and Linked Data.

II. Collaborative Filtering

CF is based on the assumption that users/items can be well predicted by other users/items that behaved in a similar way in the past. In order to put this into practice, the traditional approach is based on the *nearest-neighbor* (*k*NN) algorithm. In user-based CF, we build neighborhoods of users while in the item-based approach, neighborhoods are created from similar items. In both cases, though, in order to compute nearest neighbors, we need a previous step of computing similarities between all users or all items. Regardless of the particular similarity measure – cosine and Pearson correlation are among the most used –, this is a costly operation that is computed in a centralized manner, since the distance computation needs to access all pairs of users or items. The formation of neighborhoods, or at least the computation of the similarity matrix, is usually an *off-line* process that, similarly to the index generation in a search engine, is repeated at regular time intervals.

Once we have obtained the k neighbors $N = n_1...n_k$ for a target user u, we can predict the estimated rating value of a given item i by computing a similarity-weighted average of the ratings from each neighbor n in N [20]:

$$r_{ui} = \sigma_u + \frac{\sum_{n \subseteq N} (r_{ni} - \sigma_n) sim(n, a)}{\sum sim(n, a)} \quad (1)$$

where r_{ui} is the predicted rating of item i for user u, r_{ni} is the known rating for user n to item i, and σ_u and σ_n are the respective mean ratings.

A. Practical limitations of CF

Despite the robustness of the CF techniques and popularity for building Web-based recommendation systems [1], the nature of the CF algorithm, which strongly depends on the user ratings, can lead to several problems. We summarize some of the most prominent issues in the context of this paper in the following paragraphs.

1) Noise and malicious feedback: Users introduce noise when giving their feedback to a recommender system, both in the form of careless ratings (*i.e. Natural Noise*) [15] and malicious entries [9], [14], [21]. Due to the nature of CF systems relying on the user ratings to generate the recommendation list, this noise can seriously harm the performance. In previous work, we have shown that inconsistencies in user ratings can negatively impact the quality of the prediction that would be given by a recommender system [3].

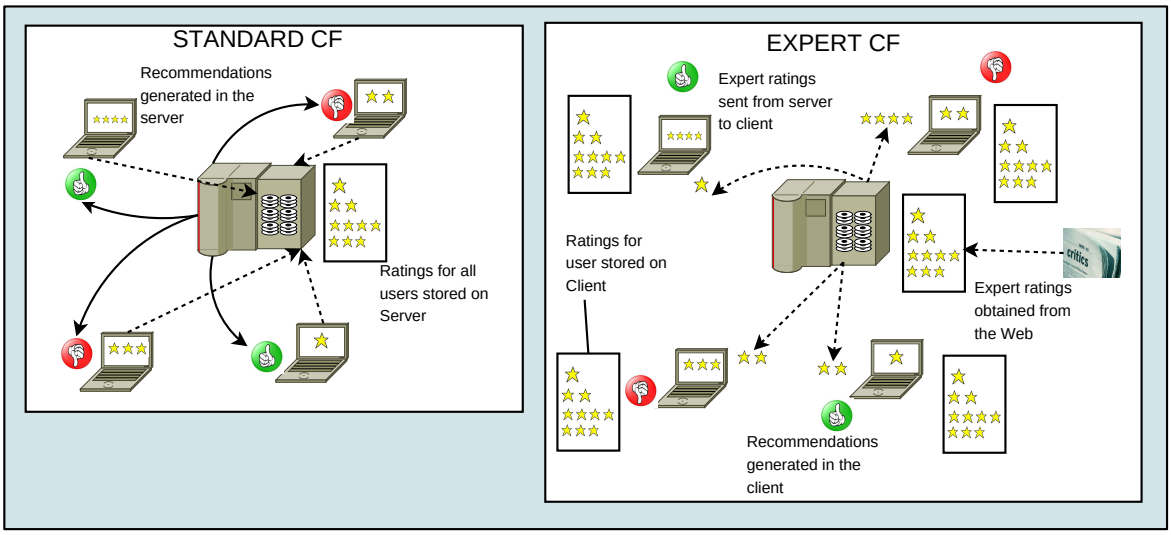

Fig. 1. Comparison of standard CF and Expert CF.

2) Sparseness: In a standard collaborative recommender system, the user-rating data is very sparse. Although dimensionality reduction techniques such as matrix factorization [24] can offer some help, this problem is still a source of inconsistency and error in the predictions.

3) Cold-start: In a CF system, new items lack rating data and can not be recommended; the same is true when a new user enters the system [12], [16]. This poses an important problem on CF recommender systems that usually need to fall back on other methods such as content-based approaches in these situations.

4) Scalability: Computing the similarity matrix for N users in an M-item collection is an $O(N^2M)$ problem. This matrix needs to be updated on a regular basis, as new items and/or users enter the system. Therefore, CF based approaches typically suffer from scalability limitations. While there are several ways to address this issue – such as k-means clustering [25] –, scalability is still an open research problem in CF systems.

5) Privacy Issues: Privacy in CF recommender systems is a growing concern and still an area of research [5], [23]. Conventional CF algorithms store all user profiles in a central location to aggregate everybody's information and to construct the user-item matrices. The service provider needs to know past ratings of the target users and their peers in order to execute the algorithms. However, this can pose a serious threat to the privacy of the users, because their personal interest or preferences can be obtained from the profiles.

III. Expert Collaborative Filtering

In our previous work [2], we presented a new approach called *expert collaborative filtering* that can tackle these problems. Basically, it replaces the peer user ratings with expert ratings and implements the collaborative filtering solution using only the experts opinions.

The *expert-based* approach for CF uses expert ratings instead of the peers'. All expert ratings for a specific domain are crawled and collected. This list of ratings can be downloaded anytime to the clients and be used for completing the calculations required for CF. Figure 1 illustrates the main components in expert CF as compared to standard CF. We can summarize the approach as follows.

(1) Crawl available expert ratings on items and build expert-item ratings matrix (*server side*).
(2) Download expert-item ratings matrix from the server (*to client*).
(3) Get user feedbacks and store them into local user profiles (*client*).
(4) Compare the user profile and the matrix and calculate k-Nearest experts (*client*).
(5) Calculate recommendation list from the ratings of the neighbor experts (*client*).

Our approach does not require the user-user similarity to be computed; instead, we build a similarity matrix between each user and the expert set. In order to predict a user's rating for a particular item, we look for the experts whose similarity to the given user is greater than δ. Formally: given a space V of users and experts and a similarity measure $sim\colon V \times V \rightarrow \mathbb{R}$, we define a set of experts $E = \{e_1, ..., e_k\} \subseteq V$ and a set of users $U = \{u_1, ..., u_N\} \subseteq V$. Given a particular user $u \subseteq U$ and a value δ, we find the set of experts $E' \subseteq E$ such that: $\forall e \subseteq E' \Rightarrow sim(u, e) \geq \delta$.

We also define the *confidence threshold* τ as the *minimum* number of expert neighbors who must have rated the item in order to trust their prediction. Given the set of experts E' found in the previous step and an item i, we find the subset $E'' = e_1...e_n \subseteq E'$ such that $\forall e \subseteq E'' \Rightarrow r_{ei} \neq \circ$, where r_{ei} is the rating of item i by expert $e \subseteq E'$, and $\circ$ is the value of the *unrated item*. If $n < \tau$, no prediction can be made and the user mean is returned, else a predicted rating can be computed and this is done using a simple variation of Eq. 1 where we replace neighbors $N = n_1...n_k$ by experts $E'' = e_1...e_n$:

$$r_{ui} = \sigma_u + \frac{\sum_{e \subseteq E''} (r_{ui} - \sigma_e) sim(e, a)}{\sum sim(e, a)} \quad (2)$$

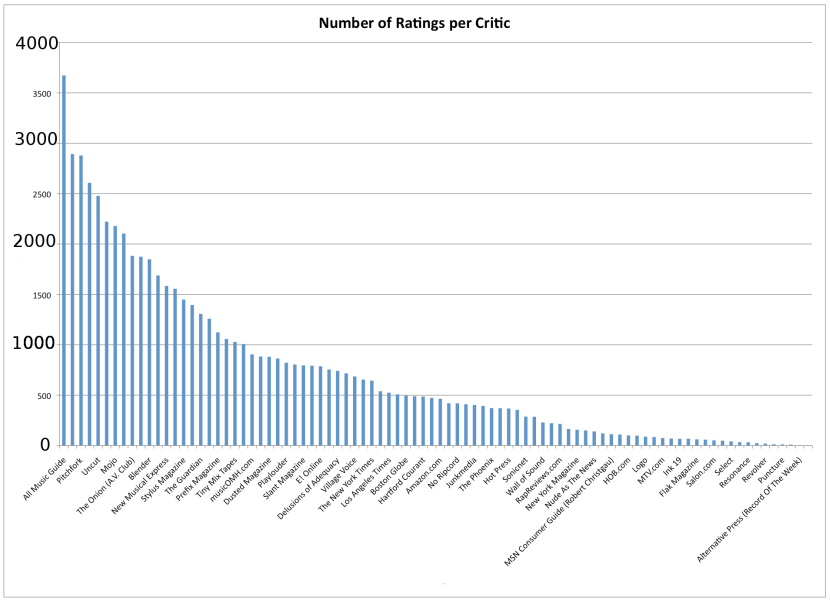

Fig. 2. Number of ratings per critic

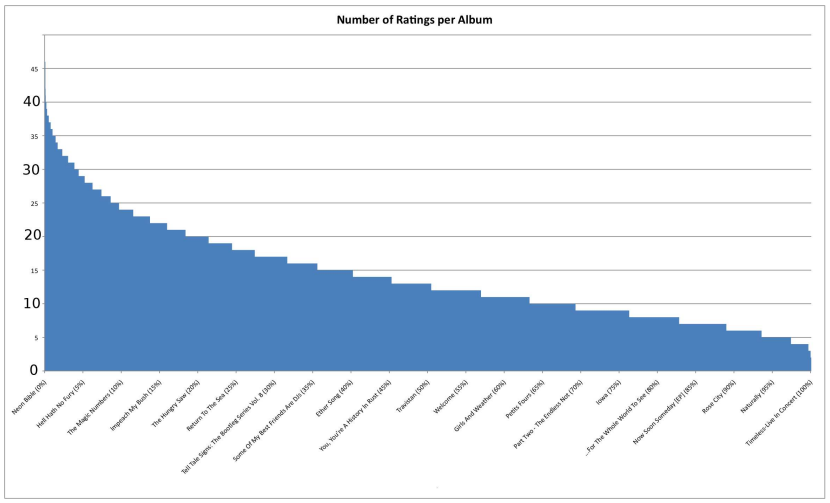

Fig. 3. Number of ratings per album

Our approach addresses some of the traditional shortcomings reviewed in the previous section. Ratings coming from authoritative experts in a given domain are much less likely to suffer from **natural noise**. Therefore, and as shown in our previous work [4], we expect ratings with less natural noise to produce more accurate recommendations. As a matter of fact, in our previous study [2], we explored the quality and the prediction power of the expert-CF algorithm and found its potential as a noise-free and competitive recommendation alternative. Also, because we are using ratings only coming from a small and supervised pool of experts, we will avoid any possibility that **malicious** ratings are injected into our datasets.

We expect experts to have a professional incentive to rate items as soon as they appear. On the one hand, this means that, on average, they will rate many more items than a regular user therefore minimizing **sparsity** of the rating matrix. On the other hand, this will also help minimize the problem of the item **cold-start** since we expect to have ratings even before any user has access to the content.

The **scalability** problem stems from the great size of the user pool that need to be considered in usual CF approaches. The number of users can grow up to millions or billions depending on the size of the service subscribers. However, we can achieve quality ratings and recommendations from a relatively reduced number of raters.

A. *Expert CF as a Privacy-preserving CF*

Although expert CF builds upon traditional CF algorithms, its dataflow is completely different, since every process except crawling the expert ratings can be done at the client side (see Fig. 1). User feedback in the form of ratings does not need to leave the local client, so user privacy is fully preserved.

Our approach guarantees complete preservation of privacy for users reporting their ratings, while still offering the advantages of CF. We can accomplish this because we draw a clear

TABLE I
STATISTICS OF CRAWLED DATA

	Albums	Critics	Ratings
Count	4,484	63	62,177

TABLE II
AVERAGE NUMBER OF RATINGS

	Per Album	Per Critic
Average rating count	13.9	986.9
Standard deviation	7.2	786.0

TABLE III
RATINGS DISTRIBUTION BY CRITICS

Rank	Critic	Ratings	Cumulative ratio
1	All Music Guide	3,673	0.06
2	PopMatters	2,893	0.10
3	Pitchfork	2,879	0.15
4	Q Magazine	2,607	0.19
5	Uncut	2,478	0.23
6	Rolling Stone	2,222	0.26
7	Mojo	2,179	0.30
8	Entertainment Weekly	2,104	0.33
9	The Onion (A.V. Club)	1,882	0.36
10	Spin	1,874	0.39
⋮	⋮	⋮	⋮
46	CDNow	420	0.90
47	No Ripcord	419	0.91

distinction between the target user and its related private profile, and the public profiles from experts that can be transmitted to all clients without worrying about privacy issues. One of the reasons we can do this in our particular implementation is because we use a reduced number of experts, and it is feasible to transmit the whole rating matrix. However, note that we could implement methods for optimizing this transmission and perform it, for instance, in an incremental manner.

Traditional approaches to CF do need to somehow share user information. Therefore, approaches to preserve privacy necessarily need to use more complex models. Canny [8], for instance, proposes a privacy-preserving protocol in which user ratings are not transmitted, but only an aggregate. This is probably the most complete approach to privacy-preservation but it still has several shortcomings – such as the fact that it is vulnerable to malicious attacks and it may affect the recommendation accuracy.

Other more "traditional" approaches to privacy preservation include using randomized perturbation to add noise to the ratings so they cannot be used in isolation to infer the user preferences [18], [26].

IV. COLLECTING EXPERT RATINGS IN THE MUSIC DOMAIN

One of our goals for this study is to verify the expert CF approach in a different setting and domain. Because our original study focused on recommending movies, for this work we decided to extend it by applying the idea to the music domain. Expert ratings for music albums were collected from Metacritic.com[1], a web site that houses reviews for various media, such as movies, DVD, TV, music, and games. This is also interesting since our previous study was using ratings from a different source – RottenTomatoes [2] – so we are now also verifying our approach is not source-dependent.

Music reviews list music album metadata (album title, artist, label, genre, etc) and the ratings by music critics and plain users for each album using a 0 to 100 point scale. In order to adhere to the definition of our expert CF, we decided to only use the critics' ratings. These are mostly from specialized music magazines or Web sites (see Table III). In our previous work, we showed that filtering out critics with few ratings did not impact the recommendation accuracy. Therefore, here we also decided to remove those critics that had less than 250 ratings.

After downloading the data to a server, we stored it into a database and created the critic-item matrix which will later be exposed through a REST API (see Sec. V-A).

A. Data Analysis

Table I shows the basic statistics of the crawled data (as of July 1st, 2009). There were 4,484 albums in the entire dataset that were originally rated by 88 critics in total. However, we kept only 63 of these critics after discarding those that had less than 250 ratings, as described above. The total number of ratings were more than 62,000. Table II, shows the average number of ratings per album and critic. Each album had 13.9 ratings and critics posted almost 1000 reviews on average. However, the large standard deviation of the critics rating count (786) suggests the distribution of the ratings should be quite uneven among critics. Figure 2 corresponds to this prospect. The graph shows a very long tail where a limited number of critics are responsible for most of the ratings. The single most frequent critic (All Music Guide) generates about 6% (3,673) of all the ratings, top 10 critics rated 39% , and less than 50 (around half of the entire group) critics rated 90% of them (Table III). If we look at the distribution of ratings per album (see Figure 3), wee see that ratings are more evenly distributed.

Figure 4 depicts the CDF (cumulative distribution function) of the mean (a) and the standard deviation (b) of ratings by album. The average rating is around 70 (out of 100). Very few albums (less than 20%) have average ratings lower than 60 and about 10% of the albums rated higher than 80 on average.

The average scores by the critics (see Fig. 5) is more evenly distributed compared to the average scores by the albums, which may be disadvantageous for filtering albums because the scores among the critics could show less discrimination. However, in the higher average score region (greater than 80), the distribution is closer to the distribution of the average rating by albums, which suggests the higher discrimination power by the critics in this score area. At the same time, The critics showed more variance than the albums. The average standard deviation by the album and by the critic was 12.74

[1] http://www.metacritic.com
[2] http://www.rottentomatoes.com

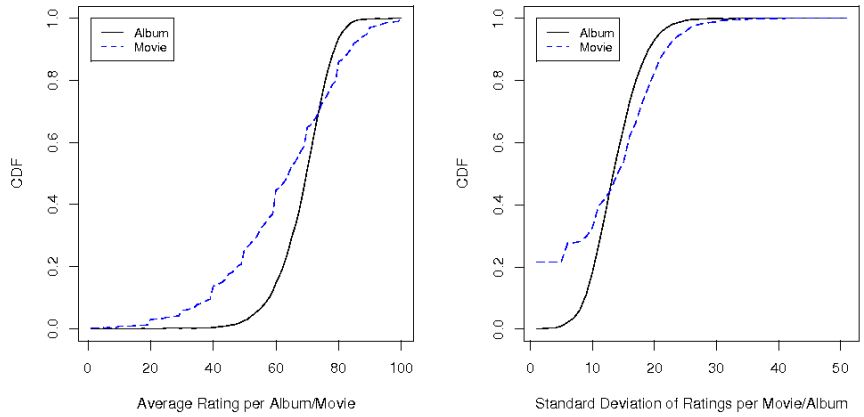

Fig. 4. Comparisons of (a) average rating and (b) rating standard deviation by album or movie for experts in the music and cinema domain

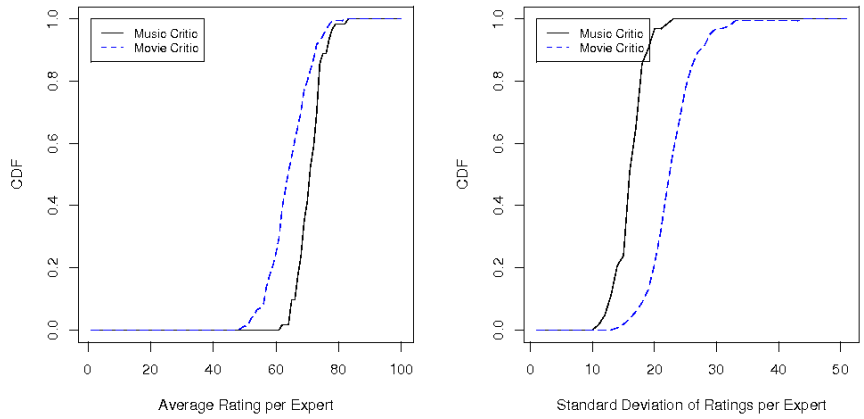

Fig. 5. Comparisons of (a) average rating and (b) rating standard deviation by music or movie critic

and 15.07 respectively. This difference suggests that the albums have more consistent rating scores and the critics were able to discriminate the quality of the albums by giving them a larger range of the scores.

Both Figure 4 and 5 also include the data for the movie critics in RottenTomatoes for comparison purposes. In Fig. 4 we see a very similar behavior in terms of std per movie and item. However, there is an important difference in how the average is distributed. Movie critics rate more movies with negative ratings (40% below 60 as compared to the 20% in albums). Our hypothesis is that while music critics tend to specialize and focus on music of a certain kind – that better suits the critic's taste – movie critics rate anything that is released. This is possible because of the smaller number of releases in the cinema domain. To support this, in Fig. 5, we see that the average per critic is slightly lower for movies than music. Furthermore, we also see that the std is much lower for music than movie critics.

B. Enhancing available information through Linked Data

In order to enhance the information available as a result of our Metacritics crawl, we make use of Linked Data resources. We use GNAT [19] to find corresponding related Linked Data identifiers for albums rated by the experts. GNAT uses the available metadata from MetaCritics to identify the songs MBID on MusicBrainz, and then outputs RDF statements representing the links to the remote web identifiers. GNAT uses a graph matching algorithm that allows for robust matching even with inaccurate or incomplete metadata.

Using the MBID and the RDF statements, we can access several Linked Open Datasets and extract high-level descriptors. At this point, we limit ourselves to the use of data obtained directly from MusicBrainz's Linked Data. However, we plan to extend this in the future to include information such as such as editorial metadata, user comments, genre, album reviews, or tags available in other Linked Open Datasets.

V. Prototype System

In this section, we describe the architecture of our proposed prototype system. The goal of our prototype is to implement a client-side expert-based CF application in the music domain. The prototype was built as a Rich Internet Application (RIA), using the most suitable technologies at each stage. Figure 6 shows the components and the dataflow of the system. The server collects the expert ratings and stores them in a

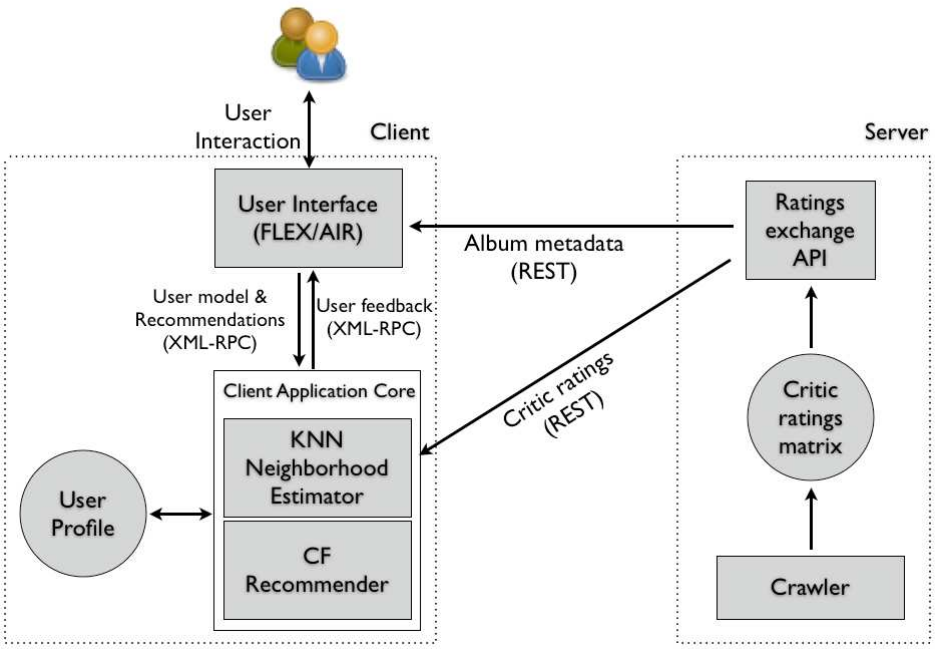

Fig. 6. Wisdom of the Few Components and Dataflow

critic-item matrix that can be used by the clients for the recommendation process. This matrix does not store any user data and it is therefore unique for all users. The user profiles where user ratings are stored, the recommendation module, as well as the user interface all reside in the client machines. In order to make the communication between the client and the server efficient, we defined an API following the REST architectural style. Within clients, the components communicate using the XML-RPC protocol. As a result of our design decisions, and in order to allow for future extensions an re-usability of the data, we developed our application as a Linked Data application [11].

A. RESTful Linked Data for Critic Ratings

In section IV-B, we explained the use of Linked Data to enhance our application. However, in order to make our application Linked Data compliant we need to observe its basic principles [6]: (1) Use URIs as names for things; (2) Use HTTP URIs so that people can look up those names; (3) When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL); and (4) Include links to other URIs. so that they can discover more things. We shall now explain a RESTful API that exposes our experts rating data while linking it to other datasets and observing the Linked Data principles.

We define an API for the client-server communication using the REST (Representational State Transfer) style. REST is a style of software architecture for distributed hypermedia system such as the World Wide Web, emphasizing scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems [10]. Systems following the REST architecture are referred to as RESTful, and they usually have the following aspects (1) the base URI for the web service, (2) the MIME type of the data supported by the Web service, and (3) the set of operations supported by the web service using HTTP methods (GET, POST, PUT, or DELETE).

Using our RESTful API, the client modules can easily get resources from the server with unique URIs. The resource is the information about the expert ratings in this case. For example, all available services are listed from the base URI using the GET HTTP method `GET <http://SERVER-ADDR/wotf/services>.`.
Figure 7 shows some examples that can be returned from the service. As it can be seen from these examples, this architecture supports unique and consistent channels to the resources available from the server and helps to build the system scalable, modular, and independent to the client architecture. Furthermore, note that we use RDF to describe our data and link it with remote datasets.

To sum up, because we strive for **service** interoperability and extensibility, we choose to use a RESTful architecture. But, because we also target **data** interoperability and extensibility, we base our design on the Linked Data principles.

B. Local CF

The local CF module operates in two stages: (1) kNN calculation and (2) generating recommendation lists. It makes use of the critic-item rating matrix downloaded from the server using the API (section V-A). Cosine similarities between the user rating vector and expert vectors are calculated and the most similar critics to the current user are selected. In principle, this calculation needs to be re-done whenever the ratings vector is updated (either user or the critics). However, despite the smaller size of the expert-CF ratings compared to conventional CF ratings, it is still expensive to download

```
GET http://SERVER-ADDR/wotf/services

<WisdomOfTheFew>
<services>
<service>./ratings/all</service>
<service>./albums/recent</service>
<service>./albums/best</service>
<service>./albums/id</service>
</services>
</WisdomOfTheFew>

GET http://SERVER-ADDR/wotf/services/albums/id/10

<album>
<album_id>10</album_id>
<artist>Peter Broderick</artist>
<title>Home</title>
<label>Hush</label>
<release_d>23 September 2008</release_d>
<release_u>1222124400</release_u>
<discs>1 disc</discs>
<genre>Rock, Folk, Singer-Songwriter</genre>
</album>
```

Fig. 7. REST example: Service root(above), individual album information (below)

TABLE IV
RECOMMENDATION PERFORMANCE

Task	Downloading	Indexing	kNN	Filtering
Time (second)	5.704	0.495	0.711	0.886

the ratings and calculate all similarities. Therefore, we just periodically update the nearest-neighbor list and store it for the next stage. This approach can be more appropriate to a specific type of clients, such as mobile devices, where we can make use of the idle time of the machine.

In the second stage, we compute the predicted rating r_{ai} of the item i unknown to a user a using equation 2.Similarity scores are cached from the previous stage, so this part can be done on the fly whenever users update their ratings on any item.

We implemented this module in Python, which is a relatively slower script language. However, thanks to the reduced size of the expert ratings and the pre-computation in the first stage, we could achieve almost instant feedbacks from the module to the user interface. Table IV shows the performance of each stage of the whole recommendation process. Even though we didn't conduct a formal benchmark testing by strictly controlling various conditions, we believe this result is enough to show the feasibility of our approach because we just used plain hardware/settings so that we could best simulate the ordinary use of information access devices. We ran the four recommendation stages individually five times and averaged the running time. By investing enough time (around 6 seconds) for downloading and indexing the expert-item matrix from the server we could calculate the neighbors and perform the filtering quickly. Both stages spent less than one second each. Considering the neighbor selection result can be saved and reused, we can perform the filtering less than in one second, which can provide users with prompt reactions as they interact with the system.

C. User Interface

The user interface is implemented in Adobe Flex/AIR. Flex[3] is a software development kit for the development and deployment of cross-platform rich Internet applications based on the Adobe Flash platform and AIR (Adobe Integrated Runtime)[4] is a cross-platform runtime environment for Adobe Flash, Flex, HTML, or Ajax. Using the framework, we were able to build an attractive and interactive user interface easily. The application can be launched via a Web browser or run independently without a browser on desktop machines using the AIR runtime. Flex/AIR is also very appropriate for building a RIA, and is also available for some mobile platforms. The choice of this framework allows us to let our users run the application on various platforms without the hassle of porting the user interface codes. Figure 8 shows the screenshot of the interface. It has 4 tabs: Recent Releases, Top 20 Albums, Search Results, and Recommended.

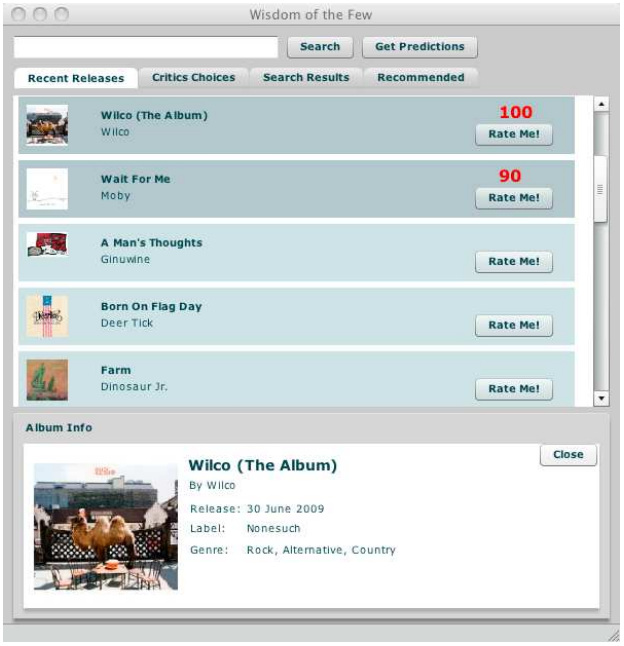

Fig. 8. Wisdom of the Few user interface showing the detailed album information

D. Communication between UI and CF algorithm

The client modules, (1) the user interface, (2) the neighborhood selection component, and (3) the rating prediction module communicate with each other using the XML-RPC protocol. XML-RPC is a remote procedure call protocol that uses XML to encode its calls and HTTP as a transport mechanism. Unlike REST, which we used for the API between the server and the client, it is a protocol rather than a style. Using XML-RPC, a software module can call procedures in a separate module, and complex objects are exchanged between the modules encoded in XML format.

We adopted XML-RPC for the client side because it is more oriented to calling procedures. We used different technologies for each components: Flex/AIR for the user interface, and expert selection module in Python and expert-CF calculation module in Python. For example, the user interface written in Flex/AIR needs to receive album ratings from users and then

[3]http://www.adobe.com/products/flex

[4]http://www.adobe.com/products/air

pass it to a module that maintains the user model. At the same time, when a user requests a recommendation list, it needs to call the recommendation module with specific arguments such as number of recommended items or the similarity threshold to sort out best matching neighbors. The recommendation module returns a list of items, which encapsulates various data fields such as album id, title, artist, and so forth. This information is decoded and incorporated to the user interface.

Therefore, considering this complex and more procedure oriented nature of the communication among client components, we decided to use XML-RPC for the client side data exchange. Compared to REST, which is more appropriate exchanging pre-computed set of data or resources, the procedure oriented approaches can maintain complex sessions, hide complex operations behind facade, and are easier to debug [10] [27].

VI. Conclusions

Traditional collaborative filtering has some well-known shortcomings such as the problem of ratings quality, user privacy, and scalability of data, which can harm the performance of the systems. An expert-based CF approach can address these problems. By exploiting the expert ratings instead of plain user ratings, we can minimize the effect of natural noise in ratings while avoiding any possibility of shilling. The use of a reduced and less sparse dataset allows for more efficient computation than the traditional CF approaches and it avoids cold-start problems.

In this paper we have shown that the approach is not limited to the movie recommendation domain and can be directly ported to the music domain. Our framework is not limited to music or movies because it is flexible and straightforward enough to be applied to any domain where expert ratings are available.

We have also proposed a client-based expert-CF architecture, which maintains the user profiles in users' machines and performs the calculation locally with the expert ratings downloaded from the server. This approach can fundamentally solve privacy issues because there is no chance that any private information is transmitted to remote machines. Besides, it is completely scalable and promises good and efficient collaborative recommendations.

Finally, we have shown how to develop this kind of RIA by combining a RESTful architectural style with Linked Data's basic principles. This will allow for reusability, no only of services but also of the data gathered and linked through the application.

Our future work includes the evaluation of the expert-CF idea and a complete user study of the prototype introduced in the paper. We also plan to deploy the prototype in a large scenario with a large enough number of users. Also, we plan on making a more extensive use of available Linked Open Data sets in order to enhance the application with related music information. Finally, we are currently working on porting the architectural solution to a mobile scenario.

Acknowledgements

This work has been partly funded by an ICREA grant from the Catalan Government.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, 2005.

[2] X. Amatriain, N. Lathia, J. M. Pujol, H. Kwak, and N. Oliver. The wisdom of the few: A collaborative filtering approach based on expert opinions from the web. In *Proc. of SIGIR '09*, 2009.

[3] X. Amatriain, J. M. Pujol, and N. Oliver. I like it... i like it not: Evaluating user ratings noise in recommender systems. In *UMAP '09*, 2009.

[4] X. Amatriain, J. M. Pujol, N. Tintarev, and N. Oliver. Rate it again: Increasing recommendation accuracy by user re-rating. In *Recys '09*, 2009.

[5] R. Baraglia, C. Lucchese, S. Orlando, M. Serrano', and F. Silvestri. A privacy preserving web recommender system. In *Proc. of SAC '06*, pages 559–563, New York, NY, USA, 2006. ACM.

[6] T. Berners-Lee. Linked data - design issues. retrieved 10-29-09.

[7] C. Bizer, T. Heath, and T. Berners-Lee. Linked data – the story so far. *International Journal on Semantic Web and Information Systems*, 2009.

[8] J. Canny. Collaborative filtering with privacy. In *IEEE 2002 Symposium on Security and Privacy*, pages 45,57, 2002.

[9] Z. C. Cheng and N. Hurley. Effective diverse and obfuscated attacks on model-based recommender systems. In *ACM Recsys '09*, page 141, 2009.

[10] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000.

[11] M. Hausenblas. Exploiting linked data to build web applications. *IEEE Internet Computing*, 2009.

[12] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong. Addressing cold-start problem in recommendation systems. In *ICUIMC '08*, pages 208–211, New York, NY, USA, 2008. ACM.

[13] P. Lisboa, M. Ling, T. Etchells, and J. Whittaker. New research directions in recommender systems for retail grocery domain. In *Workshops on Industrial Applications of Recommender Systems at Recsys '09*, 2009.

[14] B. Mehta and W. Nejdl. Attack resistant collaborative filtering. In *Proc. of SIGIR '08*, 2008.

[15] M. P. O'Mahony. Detecting noise in recommender system databases. In *Proc. of IUI'06*, pages 109–115, 2006.

[16] S.-T. Park and W. Chu. Pairwise preference regression for cold-start recommendation. In *ACM Recsys '09*, page 21, 2009.

[17] M. Perrero, F. Antonelli, and M. Geymona. Recommendation on tv: What do users want? a user study. In *Workshops on Industrial Applications of Recommender Systems at Recsys '09*, 2009.

[18] H. Polat and W. Du. Privacy-preserving collaborative filtering using randomized perturbation techniques. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 625, 2003.

[19] Y. Raimond, C. Sutton, and M. Sandler. Automatic interlinking of music datasets on the semantic web. In *Proc. of IDOW08*, 2008.

[20] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proc. of ACM CSCW'94*, pages 175–186, 1994.

[21] J. F. S. Zhang, Y. Ouyang and F. Makedon. Analysis of a low-dimensional linear model under recommendation attacks. In *Proc. of SIGIR '06*, 2006.

[22] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. *The Adaptive Web*, pages 291–324, 2007.

[23] R. Shokri, P. Pedarsani, G. Theodorakopoulos, and J.-P. Hubaux. Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. In *ACM Recsys '09*, page 157, 2009.

[24] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofirank. maximum margin matrix factorization for collaborative ranking. In *Proc. of NIPS'08*, 2008.

[25] G.-R. Xue, C. Lin, Q. Yang, W. Xi, H.-J. Zeng, Y. Yu, and Z. Chen. Scalable collaborative filtering using cluster-based smoothing. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 114–121, New York, NY, USA, 2005. ACM.

[26] S. Zhang, J. Ford, and F. Makedon. A privacy-preserving collaborative filtering scheme with two-way communication. In *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pages 316–323, 2006.

[27] M. zur Muehlen, J. Nickerson, and K. Swenson. Developing web services choreography standards–the case of REST vs. SOAP. *Decision Support Systems*, 40(1):9–29, 2005.