Smart Booking Without Looking: Providing Hotel Recommendations in the TripRebel Portal

Matthias Traub
Know-Center
Graz, Austria
mtraub@know-center.at

Dominik Kowald
Know-Center
Graz, Austria
dkowald@know-center.at

Emanuel Lacic
Graz University of Technology
Graz, Austria
elacic@know-center.at

Pepijn Schoen
TripRebel GmbH
Hamburg, Germany
pepijn.schoen@triprebel.com

Gernot Supp
TripRebel GmbH
Hamburg, Germany
gernot.supp@triprebel.com

Elisabeth Lex
Graz University of Technology
Graz, Austria
elisabeth.lex@tugraz.at

ABSTRACT

In this paper, we present a scalable hotel recommender system for TripRebel, a new online booking portal. On the basis of the open-source enterprise search platform Apache Solr, we developed a system architecture with Web-based services to interact with indexed data at large scale as well as to provide hotel recommendations using various state-of-the-art recommender algorithms. We demonstrate the efficiency of our system directly using the live TripRebel portal where, in its current state, hotel alternatives for a given hotel are calculated based on data gathered from the Expedia Affiliate Network (EAN).

CCS Concepts

•**Information systems** → **Recommender systems;**

Keywords

hotel recommendations; TripRebel; Apache Solr; Expedia; scalable recommender framework; collaborative filtering

1. INTRODUCTION

The new hotel booking site TripRebel (seen in Figure 1) aims to make hotel booking easier and fairer. A common problem when searching for hotels is that too many alternatives are available, causing a choice overload. To reduce the burden of finding the right location to spend the business trip or holidays, different methods are employed, with personalized hotel recommendations as one crucial part of this. However, in order to accurately model the traveler's profile, enough knowledge about the different hotel attributes need to be provided, so that the best hotel for that specific person can be recommended. Besides focusing on the standard problem of modeling the user profile and finding the best suited hotels, another important task to consider is the current context of the user, e.g., the location of the hotel that is currently viewed. By considering the user's current context, the performance of personalized recommendations can be highly improved. For example, Cremonesi et al. did an extensive evaluation of different recommender approaches in the domain of hotel recommendation [4]. Their evaluation shows that personalized recommender approaches have a higher and more consistent user satisfaction compared to simple most popular recommendations. Personalized recommender approaches also support users by finding cheaper hotel alternatives, which is also an objective of the TripRebel portal.

Thus, in cooperation with TripRebel, we defined use cases and several requirements for such a personalized hotel recommender system. On the basis of Apache Solr, we then developed a system architecture with Web-based services to interact with incoming user interactions at large scale as well as to provide hotel recommendations using various state-of-the-art recommender approaches. In the following sections, we show the use cases and requirements a hotel recommender system needs to handle. We also describe the recommender system we implemented and show how the different hotel recommender scenarios are supported by it. Lastly, we show the full functionality of TripRebel's recommender system through an interactive REST API.

2. USE CASES AND REQUIREMENTS

To build a hotel recommender system, specific use cases and requirements need to be considered. First of all, our system should be able to recommend similar hotel alternatives based on the available content attributes and location information, and further provide personalized hotel recommendations through exploiting implicit user-hotel interaction data. Considering these two use cases, we extracted four key requirements that must be supported by a hotel recommender system:

Req1. It should be possible to filter the recommended hotels (e.g., by hotel attributes, like city or hotel location).

Req2. It should be possible for the user to combine and weight the various recommendation approaches (e.g., Collaborative Filtering and Content-Based Filtering) in form of hybrid approaches.

Req3. Hotel data updates and new user-hotel interactions (i.e., implicit data) should immediately be taken into account for the calculation of recommendations.

Req4. Recommendations should be provided at large scale and in (near) real-time.

Based on these use cases and requirements, we analyzed various recommender frameworks such as MyMediaLite[1] or Apache

[1]http://mymedialite.net

ACM 978-1-4503-3721-2/15/10.
http://dx.doi.org/10.1145/2809563.2809616

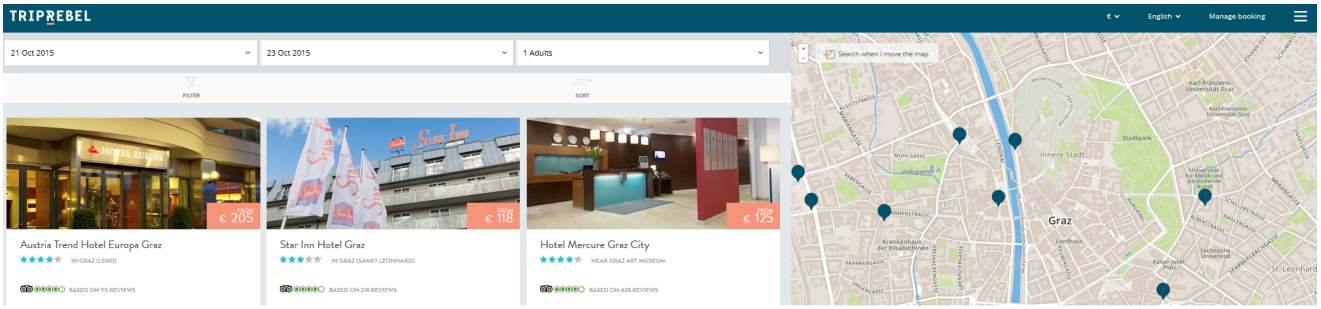

Figure 1: The TripRebel portal showing the top search results for hotels in Graz.

Mahout[2]. These frameworks are well suited when working with explicit and implicit user-item interaction data but they lack functionality to process content- and location-based data which is crucial for the domain of hotel recommendations, especially when the recommendations have to be provided in (near) real-time. For example, a content-based hotel recommender system by Levi et.al. [8] is obtaining a 10% increase of recommender accuracy compared to TripAdvisor's and Venere's top suggestions. However, a drawback of their approach is that the preprocessing of user reviews and hotel attributes clustering is not done in real-time.

Thus, we designed and developed a large scale recommender system on the basis of the open-source enterprise search platform Apache Solr which we present in detail in the next section.

3. SYSTEM OVERVIEW

In this section, we give an overview of our developed recommender system including its architecture and deployed services. The system architecture of our hotel recommender system is visualized in Figure 2. It is fully developed in Java and consists of five main components which are described in the following subsections.

3.1 Apache Solr

The data backend for our recommender system is built on Apache Solr [3]. Solr is an open-source, high-performance search engine platform providing a powerful Java-based API and a convenient user interface for easy interactions with the data (also depicted in Figure 3). We chose Apache Solr over other database solutions such as MySQL or PostgreSQL because of its intuitive query functionality, its capability of various data types (e.g., geospatial data) and its already built-in *MoreLikeThis*[4] function. With this *MoreLikeThis* functionality it is easy to process a large amount of data and still provide (near) real-time performance. Furthermore, Apache Solr provides the capability for horizontal scaling, allowing to create either shards (i.e., splitting the data into smaller indices to increase the performance of search queries for huge data sets) or replicas (i.e., cloning the existing shards to another machine to increase the fault tolerance of the whole system). Furthermore, it has already proved its usefulness for recommender systems in terms of scalability in some of our previous work (see e.g., [6, 7]).

Since the TripRebel portal currently uses the API of the Expedia Affiliate Network (EAN) and in order to get initial hotel data for our recommender system, we analyzed the freely available EAN database[5] and loaded the necessary data into our Solr index. Based on the conducted analysis and the requirements listed in Section 2, we defined four Solr cores (i.e., a running instance of an index) to be used by the recommender approaches:

Hotels. All hotel data (139, 213 hotels initially) with up to 26 different metadata attributes (e.g., name, address, description, latitude/longitude location, list of amenities, etc.).

Rooms. All room data (970, 584 rooms initially) with related information (e.g., room id, name, description and the corresponding hotel id).

UserInteractions. All interaction data between users and hotels (i.e., hotel views, likes and bookings) that happen in the TripRebel portal.

Recommendations. This core logs the metadata of each recommendation request (i.e., the input parameters, timestamp and list of recommended hotels) and is used for future evaluations, as well as tunings of the system (see Section 5).

3.2 Data Modification Layer

The Data Modification Layer (DML) component's main role is to communicate with Apache Solr (i.e., store and query data). Another responsibility of the DML component is to map the Solr specific data types to the internal ones used by the recommender approaches. We decided on a separate component to communicate with the data storage backend in order to be more flexible in switching to another technology, if we find one that would be more suitable for handling geolocation data and support real-time recommendations.

3.3 Recommender Engine

The Recommender Engine (RE) component consists of four different algorithms for generating hotel recommendations: (1) *User-Based Collaborative Filtering* (CF), (2) *Content-Based Filtering* (CBF), (4) *Most Popular* (MP), and (4) a *Hybrid* approach.

Most Popular (MP). The MP approach is not personalized and recommends the same set of hotels to every user. The resulting hotels are weighted and ranked by the frequency of bookings, likes and views.

User-Based Collaborative Filtering (CF). The basic idea of CF is that users with a similar taste (e.g., users who liked the same hotels in the past), will likely agree also on other hotels in the future [11]. Our implementation of CF is a user-based nearest neighbor algorithm which basically consists of two steps. First, the k most similar users for the target user u are found and second, each hotel h of these users is ranked using the following formula (adapted from [11]):

$$pred(u,h) = \sum_{v \in neighbors_k(u,h)} sim(u,v) \qquad (1)$$

where $pred(u,h)$ is the probability that u will like h, $sim(u,v)$ is

[2]https://mahout.apache.org/users/recommender
[3]http://lucene.apache.org/solr/
[4]https://cwiki.apache.org/confluence/display/solr/MoreLikeThis
[5]http://developer.ean.com/database/

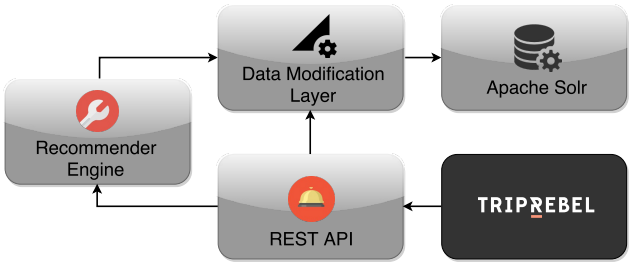

Figure 2: The software architecture of our hotel recommender system for the TripRebel portal.

the similarity between u and neighbor v and $neighbors_k(u, h)$ is the set of the k most similar users for u that have interacted with h. We define this similarity as the number of common hotel interactions which can be efficiently determined using Apache Solr.

Moreover, we distinguish between two different use cases when calling our CF approach. In the first case, a CF approach (CF_u) would be called for the current user to calculate similar users using the hotel interaction data gathered up till the point of requesting the recommendation. In the second case, we consider the current context of the user, i.e., the current hotel being viewed on the portal ($CF_{u,h}$). In this situation, we only consider similar users that have either viewed, liked or booked that exact hotel.

Content-Based Filtering (CBF). CBF approaches [1, 10] usually find similar items based on various types of metadata tied to items or users. Our approach analyzes hotel metadata to identify hotels that could be of interest to a specific user or hotel alternatives for a specific hotel. As Apache Solr stores the hotel data in a Vector Space Model (VSM), our internal API queries the data using the built-in TF-IDF ranking equation [6] to find similar hotels.

Hybrid. All of the mentioned recommender approaches have unique strengths and weaknesses (e.g., CF suffers from sparse data and cold start problems while content-based approaches suffer from item metadata to be utilized [2]). To tackle this issue and to produce more relevant recommendations, the different approaches can be combined into a hybrid recommender [3, 5, 9]. In order to realize this in an easy way, we chose to implement and modify the Mixed Hybrid approach defined in related work [2].

In the basic Mixed Hybrid approach, recommendations of each algorithm are ranked and then the top-N are picked from each source, one recommendation at a time by alternating the used algorithms. Thus, relative positions in a ranked list are considered but the importance of each individual recommendation approach is not considered. To cope with the former issue, we modified the Mixed Hybrid to let the TripRebel portal first define the importance of every approach, i.e., assign the number of recommendations each approach should return. The default Mixed Hybrid is then applied on each recommendation set. If the number of requested recommendations was higher than the resulting recommendations, the remaining spots are filled with the results of the MP approach.

3.4 REST API and TripRebel Portal

The direct access to the internal components is restricted to admin users for security and data integrity reasons. Hence, the communication between the TripRebel portal and the Recommender System goes through a separate REST API which acts as a proxy for the client's incoming HTTP requests. An example of our interactive REST API based on Swagger[7] is given in Figure 5. The

[6]https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

[7]http://swagger.io/

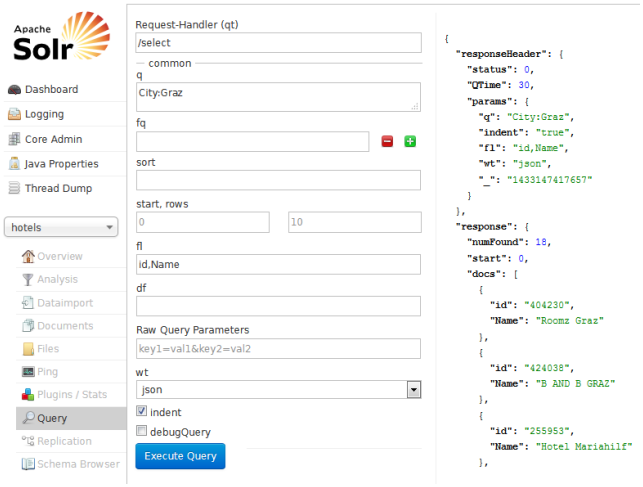

Figure 3: Graphical user interface to Apache Solr showing the query to get hotels (by id and name parameters) in Graz.

REST API consists of basic data managing functionalities used to maintain hotel related information. Besides that, all previously described recommender approaches are represented with specific calls with certain sets of parameters that can be modified for different use cases. Furthermore, all user interactions gathered from the TripRebel portal are forwarded to Apache Solr via the REST API.

4. DEMONSTRATION SCENARIO

The demonstration at the conference will be split in two parts: (i) we will show the TripRebel portal and how hotel recommendations are currently used there and (ii) we will demonstrate the full functionality of our recommender system via its interactive REST API.

4.1 Live Recommendations in TripRebel

This part of our demonstration can be tested by each participant of the conference on his/her own device. Imagine you want to go to the i-KNOW conference in Graz, Austria from the 21st until the 23rd of October 2015 and you are looking for a nice but also affordable hotel in the city. By entering your parameters into the TripRebel portal search results similar to Figure 1 will be received. Then you click on the first search result[8] which in our case is the Austria Trend Hotel Europa Graz[9]. Since this hotel seems not to be the cheapest option in Graz, TripRebel offers a "Similar Hotels" recommendation area at the end of the hotel page with some alternatives as shown in Figure 4. These suggestions are similar to the source hotel in terms of the location, description text and list of amenities but provide a better price or have been booked by similar session users.

4.2 Interactive REST API Calls

In addition to the live recommender demonstration described in the last section, we will also provide a full overview of the functionality of our recommender system via its interactive REST API. This part of the demonstration will be shown on our laptop since the full API is not publicly available.

[8]*Please note*: TripRebel is a live system in which search and recommendation results might differ if underlying data changes.

[9]http://www.triprebel.com/hotels/Graz/Austria/10-21-2015/10-23-2015/1/143524

Approach	Number of Recommendations	User Interactions related to Recommendations	Liked	Booked	Conversion Rate	Duration (min)			
						<1	1-2	2-3	>3
$CF_{u,h}$	582	130	57	25	4.29%	17	31	13	69
CF_u	401	39	15	10	2.49%	5	16	8	10
Both	983	169	72	35	3.56%	22	47	21	79

Table 1: Preliminary evaluation results for the two CF approaches, $CF_{u,h}$ (context-aware) and CF_u.

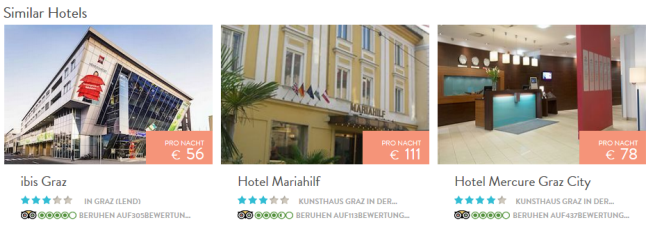

Figure 4: Current implementation of the recommender services in the TripRebel portals showing similar hotel alternatives for the Austria Trend Hotel Europa Graz.

Figure 5 shows an example of an API call visualized via Swagger. Swagger allows to enter all parameters in a convenient way via well-known web-form fields and triggering the REST call using these parameters via the "Try it out" button. In the visualized example, the service provides similar hotels for a target hotel-id and the JSON response of this call is shown in the "Response Body" box. Moreover, Swagger also provides the URL of the full REST path with all parameters in the "Request URL" field which helps the developer to integrate the service into a client application. In addition to this testing functionality, Swagger also acts as a full documentation of the API calls.

During the conference demo session, we will not only show this example but also other calls mentioned in Section 3.4. This would allow us for example, to demonstrate a scenario where we first add some user interaction into the system and then show how this new data changes the behavior of the recommender algorithm.

4.3 Preliminary Evaluation Results

In the current state of the TripRebel portal, we are comparing the two Collaborative Filtering approaches with respect to user acceptance. Thus, we present some preliminary results of this anlysis in Table 3.4. These premature results show that the $CF_{u,h}$ approach (that takes the currently viewed hotel into account) leads to more user interactions compared to CF_u (that only takes the target user into account). The 4.29% conversion rate of $CF_{u,h}$ based on the number of recommendations exceeds and almost doubles the rate of CF_u with 2.49%. These results are interesting but also expected, as they suggest that considering the current context of the user results in more accurate recommendations.

5. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel recommender system used by the hotel booking portal TripRebel. By using Apache Solr, the system can immediately process and incorporate hotel and user interaction data in real-time. Additionally, we defined and showed how to apply specific use cases and requirements for a hotel recommender system. For future work, we plan to incorporate the remaining MP, CBF and Hybrid approaches in the TripRebel portal and evaluate them there. Furthermore, we will also focus on improving the hybrid approach by trying out different strategies on letting the user dynamically provide weights and set search parameters for the recommended hotels.

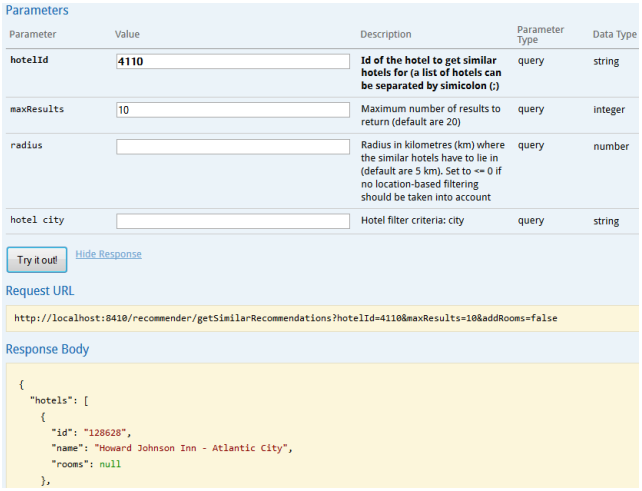

Figure 5: Swagger documentation and testing interface for one of our recommender services.

Acknowledgements. This work is supported by the Know-Center and the EU-funded project Learning Layers (Grant Agreement : 318209). The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Ministry of Transport, Innovation and Technology, the Austrian Ministry of Economics and Labor and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency (FFG). TripRebel is supported by funds of InnoRampUp, IFB Innovationsstarter, City of Hamburg, Germany.

6. REFERENCES

[1] M. Balabanović and Y. Shoham. Fab: content-based, collaborative recommendation. *Communication of ACM*, 40(3):66–72, Mar. 1997.

[2] S. Bostandjiev, J. O'Donovan, and T. Höllerer. Tasteweights: A visual interactive hybrid recommender system. In *Proc. of RecSys '12*.

[3] R. Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[4] P. Cremonesi, F. Garzotto, and M. Quadrana. Evaluating top-n recommendations "when the best are gone". In *Proc. of RecSys '13*.

[5] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proc. of RecSys '10*.

[6] E. Lacic, D. Kowald, D. Parra, M. Kahr, and C. Trattner. Towards a scalable social recommender engine for online marketplaces: The case of apache solr. In *Proc. of WWW '14*, pages 817–822, 2014.

[7] E. Lacic, D. Kowald, and C. Trattner. Socrecm: A scalable social recommender engine for online marketplaces. In *Proc. of HT '14*.

[8] A. Levi, O. Mokryn, C. Diot, and N. Taft. Finding a needle in a haystack of reviews: Cold start context-based hotel recommender system. In *Proc. of RecSys '12*.

[9] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proc. of WSDM '11*.

[10] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[11] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen. The adaptive web. chapter Collaborative Filtering Recommender Systems. 2007.