Tailoring Recommendations for a Multi-Domain Environment

Emanuel Lacic
Know-Center Graz
Graz, Austria
elacic@know-center.at

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

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

ABSTRACT

Recommender systems are acknowledged as an essential instrument to support users in finding relevant information. However, the adaptation of recommender systems to multiple domain-specific requirements and data models still remains an open challenge. In the present paper, we contribute to this sparse line of research with guidance on how to design a customizable recommender system that accounts for multiple domains with heterogeneous data. Using concrete showcase examples, we demonstrate how to setup a multi-domain system on the item and system level, and we report evaluation results for the domains of (i) LastFM, (ii) FourSquare, and (iii) MovieLens. We believe that our findings and guidelines can support developers and researchers of recommender systems to easily adapt and deploy a recommender system in distributed environments, as well as to develop and evaluate algorithms suited for multi-domain settings.

KEYWORDS

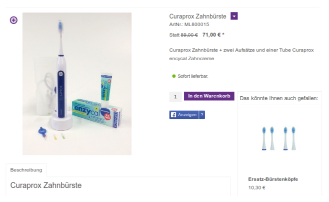
Recommender Systems; Multi-Domain Recommendation; Heterogeneous Data; Customizing Recommendation Approaches

1 INTRODUCTION

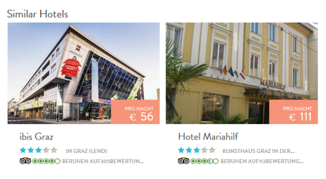
In the past decade, there has been a vast amount of research in the field of recommender systems. Most of these systems offer recommendations adapted for items belonging to a single domain (e.g., movies , music , news , etc.). However, supporting different domain-specific data models is still an open challenge, which is neglected by many recommender systems and that has just been recently taken up by the recommender systems' research community.

The work of [2] has actually been the first attempt to define the concept of a domain in the context of recommender systems. The authors distinguish between four different domain notations, namely (i) attribute level, where items are of the same type (e.g., movie genres), (ii) type level, where items are of similar types but share some attributes (e.g., movies and tv shows), (iii) item level, where items are not of the same type and differ in most or all attributes (e.g., movies and books) and, (iv) system level, where items and users belong to different systems (e.g., LastFM and MovieLens). Moreover, they distinguish between the concept of multi-domain recommendations and cross-domain recommendations. The goal of cross-domain recommendation is to utilize the knowledge derived

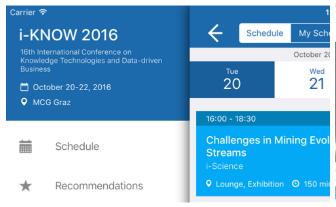
RecSysKTL Workshop @ ACM RecSys '17, August 27, 2017, Como, Italy



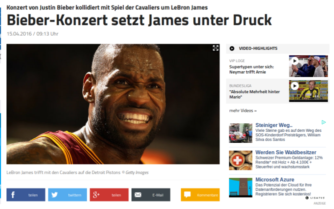
(a) E-commerce.



(b) Hotels.



(c) Scientific talks.



(d) News articles.

Figure 1: Providing recommendations in multiple domains requires supporting heterogeneous data structures, allow for domain-specific algorithm customization, as well as to support service isolation and fault tolerance.

from one or more source domains in order to generate predictions for a target domain. However, they also state that multi-domain approaches have mainly focused on the provision of cross-domain recommendations by jointly considering user preferences for items in various systems.

Other related works implicitly share this definition [2] and focus on cross-domain recommendations rather than on multi-domain ones. Thus, the main focus has been to tackle the data sparsity problem (e.g., via active transfer learning [14]) by utilizing Collaborative Filtering [4, 7, 10] and Content-based Filtering approaches [3, 11].

In the present paper, we build upon these works and we extend the scope of multi-domain recommender systems by introducing several guidelines concerning topics such as data heterogeneity and customization that should be taken into consideration. We base our findings on real-world applications (e.g., e-commerce[1], hotels[2], scientific talks[3] or news articles[4] – just to name a few) and propose a practical approach on how to support muti-domain recommendations on both the item and system level as described by [2].

To the best of our knowledge, this is the first work which addresses the question of what design decisions should be taken into consideration when building a recommender system for a multi-domain environment.

[1] http://www.mymanou.com/
[2] https://www.triprebel.com/
[3] http://uscn.me/rr220
[4] http://www.clef-newsreel.org/

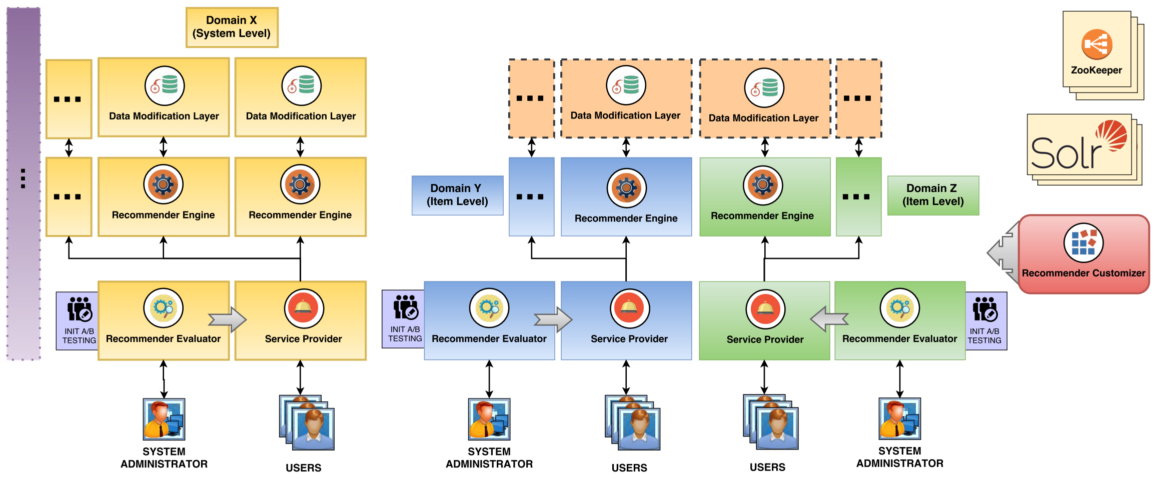


Figure 2: Proposed system architecture of a multi-domain recommendation environment showing how the various modules work together. Each module is a standalone HTTP server, which is aware of the location (i.e., URL) of its communicating partners. In case of an item level multi-domain scenario, the same data storage is shared between the domains and customization via recommender profiles provides the domain-specific algorithm configuration.

2 A MULTI-DOMAIN RECOMMENDATION APPROACH

In this section, we categorize and propose guidelines to extend the notation of a multi-domain recommender system. We base our guidelines on our previous work [5, 6, 12], which we have already applied in live settings in various domains (e.g., e-commerce, hotels, conference or news as shown in Figure 1).

Thus, we propose four issues that should be addressed when providing multi-domain recommendation, i.e., (i) service isolation, (ii) data heterogeneity, (iii) recommender customization, and (iv) fault tolerance. While we focus on item and system level domain notations, our findings can be adapted for both the attribute and type level by means of additional recommender customization (e.g., filtering by the item category).

2.1 Service Isolation

When supporting multi-domain recommendations on the system level, effective hardware utilization is crucial. Actually, as each domain has different requirements with respect to the request load, the hardware utilization rate can be improved by sharing the same hardware resources across multiple domains. Additionally, in such a scenario, performance isolation is crucial. Specifically, a high request load in combination with possible performance-intensive operations needed for one domain should not impact the performance in another domain. For example, news recommender systems usually have a requirement of providing session-based recommendations within 100 milliseconds and need to cope with challenging load peaks during morning hours and the lunch break at working days [13]. Thus, in cases where the request load is too large for a particular domain, it should be possible to dynamically scale the system and handle such performance intensive load peaks.

Correspondingly, we propose to separate a recommender system's logic into several microservices by adopting the Microservices architecture design pattern[5]. An example of such an architecture is shown in Figure 2. Here, five different modules take care of (i) data handling, (ii) calculating recommendations, (iii) balancing incoming recommendation requests, (iv) domain-specific customization, and (v) evaluating recommendations. To support horizontal scaling and to coordinate all deployed modules, as well as the corresponding system level domain assignments, we use Apache ZooKeeper[6]. This overall approach can be extended with virtualization and container technologies such as Docker[7] or LXC[8]. These lightweight resource containers provide features such as portability, more efficient scheduling and resource management, as well as less virtualization overhead, which are beneficial when implementing a multi-domain recommender on the system level.

2.2 Heterogeneous Data

As the amount of data is doubled approximately every 40 months [8], most recommender systems migrate from traditional databases to distributed systems that can scale more easily and handle massive streams of heterogeneous data. As such, a multi-domain recommender system needs to handle a diverse set of data (e.g., ratings, views, likes, check-ins, etc.), while at the same time enabling an easy integration of new types by modifying the underlying schema.

In our case, we leverage the Apache Solr search engine and found its schema-less mode[9] to be a great fit to support multi-domain recommendations on an item level, as it allows dynamic schema construction by indexing data without the need to edit it manually.

[5]http://microservices.io/patterns/microservices.html
[6]http://zookeeper.apache.org/
[7]http://www.docker.com
[8]https://linuxcontainers.org/
[9]https://cwiki.apache.org/confluence/display/solr/Schemaless+Mode

```
{
  "id": "e12c4fb−ba85−46d5−896d−af65d1f3b48c",
  "item": "5a2bc423−15dc−47d3−8a8c−f543dc267a7c",
  "users_listened": [3907, 57017]
  "users_listened_count": 2
  "domain": "LastFM"
},
{
  "id": "8fe89bab−9631−4e87−81a0−a8dd3ffba774c",
  "user": "23861",
  "item": 4105,
  "rating": 1.0
  "domain": "MovieLens"
}
```

Listing 1: Example of different schema strategies to store data in the same place for item level multi-domain recommendations.

For example, as shown in Listing 1, we could easily derive different data structures to store and generate recommendations in the corresponding music and movie domain. Moreover, we found that the capability for horizontal scaling (i.e., creating shards and replicas) is extremely important when integrating new domains on an item-level (see Figure 2) as such a strategy easily increases the amount of data that needs to be stored and processed.

2.3 Customizing Recommendation Approaches

An important aspect of a multi-domain recommender system is customization. Typically, different application domains have different domain-specific data features, which means that, for example, a music recommender could solely use implicit user interactions (e.g., listened songs), whereas an e-commerce recommender could use explicit ones (e.g., ratings). On top of that, one would also need to separately determine and setup the correct algorithmic parameters for each domain. For example, in case of the Collaborative Filtering algorithm, the similarity function (e.g., Cosine or Jaccard similarity) and the neighbourhood size need to be determined for each domain. For other domains, one may need to setup custom filtering criteria (e.g., recommend items that are suited for minors or are part of a specific category). Thus, a multi-domain recommendation approach should be aware of the underlying data structures and domain-specific parameters.

As such, we propose to outsource the domain-specific algorithm setup into so-called recommender profiles. In our applications, we defined a customizer module (see Figure 2), which enables to set domain-specific algorithm configurations and to transfer it to modules utilized in a specific domain environment. This way, we can manage domain-specific configurations and dynamically integrate additional domains. An example of such recommender profiles for the music domain is given in Listings 2, 3 and 4.

Here, we define a configuration by a unique reference id, a reference to the algorithm implementation (e.g., class name) and the specific domain-relevant parameters. In case a recommender profile is created or updated for a particular domain at runtime, the changes need to be propagated throughout the whole system to every domain-dependant module. As such, each module from the same domain will be informed about the changes and the updated profile will be used as soon as a new recommendation request is received.

```
id: ktl_mp_lastfm
# reference for a MP implementation
algorithm: MostPopularGeneric

# algorithm specific parameters
parameters:
  # item−level domain?
  domain: lastFM
  # on what do I calculate the popularity ?
   count_fields : [ users_listened_count ]
    user_action_fields : [ users_listened ]
```

Listing 2: A MostPopular recommender profile for the LastFM domain.

```
id: ktl_ub_cf_lastfm
# reference for a user−based−CF implementation
algorithm: GenericUBCF

# algorithm specific parameters
parameters:
  # item−level domain?
   domain: lastFM
  # How to calculate user similarity ?
   similarity_function : OVERLAP # JACCARD, COSINE, etc.
   neighbourhood_size: 40
   user_action_fields : [ users_listened ]
```

Listing 3: A Collaborative Filtering recommender profile for the LastFM domain.

```
id: ktl_hybrid_cs_lastfm
# reference for a hybrid implementation
algorithm: CrossSourceHybrid

# algorithm specific parameters
parameters:
  # item−level domain given by combining profiles
  profile_ids : [ ktl_mp_lastfm , ktl_ub_cf_lastfm ]
  recommender_weights: [0.1, 0.9]
```

Listing 4: A Cross-Source Hybrid recommender profile for the LastFM domain.

2.4 Fault Tolerance

Deploying multiple modules in a distributed manner increases the probability of unexpected behaviour (e.g., hardware shutdown, I/O problems, software bugs, etc.). As we have proposed to use microservices, it is not necessary to cope with central node failures as it is the case with a master-slave architecture. In case a module fails, ZooKeeper, or any other orchestration service like Eureka or Consul, should remove the faulty module from its list of "live nodes" and no further requests will be redirected to it.

Thus, the module will not necessarily cause any major problems as long as there is another module of the same type available. When experiencing a high request load, it should be possible to deploy and register an additional module to ZooKeeper on the fly. To further improve the reliability of the system, multiple ZooKeeper instances can be used in a cluster in order to overcome the outage of single instances. In such a way, the runtime performance can be guaranteed for both item and system level multi-domain recommendations.

3 DOMAIN EXPERIMENTS

In order to demonstrate the application of our guidelines for providing recommendations in multiple domains, we performed several experiments on well-known datasets used in recommender systems research, (i.e., LastFM[10], Foursquare[11] and MovieLens20M[12]). The LastFM dataset consists of 359,348 users, 268,736 artists and 17,559,530 implicit user interactions that denote the listening relationship between users and artists. Foursquare provides 2,809,581 ratings on a 5-star scale for different venues (i.e., restaurants) and contains 2,153,471 users as well as 1,143,092 venues in general. The MovieLens20M dataset has 138,493 users, 27,032 movies as well as 19,999,603 movie reviews on a 5-star rating scale with a step size of 0.5 (i.e., a 10-star scale).

We focused on providing recommendations for cold-start users and as such, we removed all users that interacted with more than 20 items. Next, we split the remaining data in two different sets (i.e., training and test sets) using a method similar to the one described in [9]. Thus, for each user, we withheld 10 items that were used for testing and the rest was used for training. This has resulted in an evaluation set of 2,409 users for LastFM, 41,628 users for FourSquare and 4,486 users for the MovieLens20M dataset. On these datasets, we evaluated a simple MostPopular (MP) approach (e.g., Listing 2), a user-based Collaborative Filtering (CF) approach (e.g., Listing 3) as well as a hybrid combination [1] of both (e.g., Listing 4). Additionally, we evaluated different neighborhood sizes N for the CF approach in order to optimize the results. For reasons of simplicity, we used a naive item overlap metric to measure the similarity between users (i.e., $OV(u_t, u_c) = |\Delta(u_t) \cap \Delta(u_c)|$, where $\Delta(u)$ corresponds to the set of items some user u has interacted with in the past). However, as shown in Listing 3 this is a domain-specific parameter that could be easily adapted.

The results of our evaluation are shown in Table 1 by means of the nDCG@10 and User Coverage (UC) metrics. The aim of this simple experiment was to show how different parameter setups can impact the performance in different domains. As shown, the neighbourhood size is one example of a parameter that needs to be optimized for a specific domain. By choosing the best parameter combination for the hybrid approach, we can provide more robust recommendations for all users in each domain.

	Approach		*nDCG*@10	*UC*
LastFM	MP		.0180	100%
	CF	N = 20	.1113	93.70%
		N = 30	.1129	
		N = 40	**.1135**	
		N = 50	.1120	
		N = 60	.1112	
	Hybrid		.1005	100%
Foursquare	MP		.0256	100%
	CF	N = 20	.0364	49.58%
		N = 30	.0403	
		N = 40	.0426	
		N = 50	.0440	
		N = 60	**.0452**	
	Hybrid		.0339	100%
MovieLens	MP		.0658	100%
	CF	N = 20	.0910	100%
		N = 30	.0945	
		N = 40	.0981	
		N = 50	.0965	
		N = 60	**.0984**	
	Hybrid		.0999	100%

Table 1: Evaluation results of our multi-domain experiment.

For future work, we plan to perform a more elaborate study using the proposed recommender profiles to study differences in domain-specific configurations (e.g., does a semantic relationship impact the choice of domain-specific parameters like the similarity metric or different filtering criteria?). Moreover, we plan to investigate datasets with textual contents in order to further explore how hybrid weightings may impact the performance in a specific domain with respect of not only accuracy but also diversity.

4 CONCLUSION

In this work, we presented our approach on providing recommendations in a multi-domain environment. Specifically, we introduced the concept of recommender profiles in order to customize existing algorithms with domain-specific configuration. Apart from that, we provided guidelines with respect to service isolation, heterogeneous data and fault tolerance. Finally, we provided customization examples as well as evaluation results for the domains of (i) LastFM, (ii) FourSquare, and (iii) MovieLens. We believe that our findings and proposed guidelines are of use for developers and researchers of recommender systems to tailor and develop recommendations for multi-domain and distributed environments.

Acknowledgment. This work was funded by the Horizon 2020 project MoreGrasp (643955).

REFERENCES

[1] S. Bostandjiev, J. O'Donovan, and T. Höllerer. Tasteweights: a visual interactive hybrid recommender system. In *Proc.*, RecSys '12, pages 35–42. ACM, 2012.
[2] I. Cantador, I. Fernández-Tobías, S. Berkovsky, and P. Cremonesi. Cross-domain recommender systems. In *Recommender Systems Handbook*. Springer, 2015.
[3] A. M. Elkahky, Y. Song, and X. He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proc. WWW'15*.
[4] S. Gao, H. Luo, D. Chen, S. Li, P. Gallinari, and J. Guo. Cross-domain recommendation via cluster-level latent factor model. In *Proc. of ECML-PKDD'13*.
[5] E. Lacic, D. Kowald, and C. Trattner. Socrecm: A scalable social recommender engine for online marketplaces. In *Proc. of the ACM Hypertext 2014*.
[6] E. Lacic, M. Traub, D. Kowald, and E. Lex. Scar: Towards a real-time recommender framework following the microservices architecture. In *Proc. of LSRS'15*.
[7] B. Loni, Y. Shi, M. Larson, and A. Hanjalic. Cross-domain collaborative filtering with factorization machines. In *ECIR*, pages 656–661. Springer, 2014.
[8] A. McAfee and E. Brynjolfsson. Big Data: The management revolution. *Harvard Business Review*, 90(10):60–68, 2012.
[9] D. Parra-Santander and P. Brusilovsky. Improving collaborative filtering in social tagging systems for the recommendation of scientific articles. In *Proc. of WI-IAT '10*, pages 136–142. IEEE Computer Society.
[10] S. Sahebi and P. Brusilovsky. It takes two to tango: An exploration of domain pairs for cross-domain collaborative filtering. In *Proc. of ACM RecSys 2015*.
[11] S. Sahebi and T. Walker. Content-based cross-domain recommendations using segmented models. In *CBRecSys@ RecSys*, pages 57–64, 2014.
[12] M. Traub, D. Kowald, E. Lacic, P. Schoen, G. Supp, and E. Lex. Smart booking without looking: Providing hotel recommendations in the triprebel portal. In *Proc. of i-KNOW '15*.
[13] S. Werner and A. Lommatzsch. Optimizing and evaluating stream-based news recommendation algorithms. In *CLEF (Working Notes)*, pages 813–824, 2014.
[14] L. Zhao, S. J. Pan, E. W. Xiang, E. Zhong, Z. Lu, and Q. Yang. Active transfer learning for cross-system recommendation. In *AAAI*, 2013.

[10]http://mtg.upf.edu/node/1671
[11]https://archive.org/details/201309_foursquare_dataset_umn
[12]https://grouplens.org/datasets/movielens/20m/