A Linked Data Recommender System using a Neighborhood-based Graph Kernel

Vito Claudio Ostuni[1], Tommaso Di Noia[1],
Roberto Mirizzi[2], and Eugenio Di Sciascio[1]

[1] Polytechnic University of Bari – Bari, Italy
[2] Yahoo! – Sunnivale, CA (US)
{vitoclaudio.ostuni,tommaso.dinoia,eugenio.disciascio}@poliba.it,
robertom@yahoo-inc.com

Abstract. The ultimate mission of a Recommender System (RS) is to help users discover items they might be interested in. In order to be really useful for the end-user, Content-based (CB) RSs need both to harvest as much information as possible about such items and to effectively handle it. The boom of `Linked Open Data` (`LOD`) datasets with their huge amount of semantically interrelated data is thus a great opportunity for boosting CB-RSs. In this paper we present a CB-RS that leverages `LOD` and profits from a *neighborhood-based graph kernel*. The proposed kernel is able to compute semantic item similarities by matching their local neighborhood graphs. Experimental evaluation on the `MovieLens` dataset shows that the proposed approach outperforms in terms of accuracy and novelty other competitive approaches.

1 Introduction

In personalized information access, the role played by recommender systems is growing considerably in importance. Every time we buy a product on Amazon, watch a movie on Netflix, listen to a song on Pandora, just to cite a few, their recommender systems suggest new items we could be interested in. Broadly speaking, existing technologies used to build recommendation engines fall in either of the following two categories: content-based filtering and collaborative filtering ones. In this work we focus on Content-based Recommender Systems (CB-RSs). They are based exclusively on domain knowledge to compute useful recommendations for the end-users by looking at information collected in their profile. On the one hand, when designing and developing a CB-RS, one of the biggest issues to face is the difficulty to get such knowledge. On the other hand, a common problem of this technique is the lack of novelty for recommended items.

The research presented here addresses these two problems by exploiting `Linked Open Data` to get domain knowledge and by proposing a neighborhood-based graph kernel. This is able to effectively handle the graph-based nature of the underlying `LOD` knowledge and capture the relations existing between items in order to compute accurate and novel recommendations. Once we represent the items by their local graph, that we call *item neighborhood graph*, the kernel

computes a weighted count of common entities between two item neighborhood graphs by taking into account their local structure. Finally, we use such kernel with SVM regression to learn the user model and to output a recommendation list.

Main contributions of this work are:

- mining of the semantics associated to items through their LOD-based local graph representation;
- formulation of a neighboorhood-based graph kernel for matching LOD-based item descriptions;
- improvement in accuracy and novelty with respect to existing CB-RSs that leverage LOD;

The rest of this paper is structured as follows. In Section 2 we detail our recommendation approach, specifying how we leverage Linked Open Data and defining a graph-kernel suitable for RDF data. The evaluation of our system and discussion of the results is carried out in Section 3. Then, we present relevant related work in Section 4. Finally, conclusion and future work conclude this paper.

2 Content-based Recommendation from LOD using graph kernels

A common way of computing content-based recommendations is learning a function that, for each item in the system, predicts the relevance of such item for the user. In a few words, the relevance represents the likelihood that the user is interested in that item [17]. The application of Machine Learning techniques is a typical way to accomplish such task [17]. A *top-N* item recommendation problem in a standard content-based setting is mainly split into two different tasks: (i) given a collection of items for which past user's preferences are available, learn a regression or classification model to predict the relevance associated to unknown items; (ii) eventually, according to such scores, the system recommends the most relevant items to the user.

More formally, let I be the set of items to use in the recommendation. For each user $u \in U$, we assume to have a collection of items $I_u \subset I$, with their associated relevance scores $r_{u,i}$. Depending on the system these scores can be derived from either implicit or explicit feedback. Given a training set for u defined as $T_u = \{(i, r_{u,i}) \text{ with } i \in I_u\}$, the two tasks for the *top-N* recommendation problem, in our setting, consist of:

1. learning a function $f_u : I \to \mathbb{R}$ from T_u which assigns a relevance score to the items in I;
2. using such function to predict the score associated to all the unknown items in $I \setminus I_u$, rank them and recommend the *top-N*.

Due to the underlying data model of RDF datasets, we are particularly interested in those machine learning methods that are appropriate for dealing with

objects structured as graphs. A popular class of techniques particularly suited for working with structured data are *Kernel Methods* [19]. Given two input objects i and j, defined in an input domain space D, the basic idea behind Kernel Methods is to construct a **kernel function** $k : D \times D \rightarrow \mathbb{R}$, that can be informally seen as a similarity measure between i and j. This function must satisfy $k(i, j) = \langle \phi(i), \phi(j) \rangle$ for all $i, j \in D$, where $\phi : D \rightarrow F$ is a mapping function to a inner product[1] feature space F. A kernel function defined in such a way must be symmetric and positive semidefinite in order to be a valid kernel [19]. Then, the classification or regression task involves linear convex methods based exclusively on inner products computed using the kernel in the embedding feature space. These methods provide a powerful framework for decoupling the data representation from the learning task.

In this work we define a novel graph-based kernel and adopt SVM Regression [19] as kernel-based algorithm for learning the user model f_u. We formalize our *top-N* item recommendation problem in a regression setting because a continuous relevance score is needed for computing the final ranking.

Hereafter we describe the way we represent items in I by means of `LOD` datasets and then we show how such a representation is leveraged in our graph-based kernel.

2.1 Graph-based Item Representation

Defining an expressive and efficient kernel starting from `RDF` data is not a trivial task. In the last few years various graph kernel functions have been designed to capture the intrinsic similarity of graphs. Unfortunately many complex kernels, such as the ones based on subgraph isomorphism, maximum common subgraphs, or graph edit distance are appropriate for chemical structures and biological networks but are not suitable for knowledge graphs. Mostly because of the size of those graphs that require computational efficient methods and because they have different properties due to the fact that they are not governed by physical laws [9]. In addition, as pointed out by [11], graphs representing chemical compounds usually have few node labels which occur frequently in the graph and nodes in these graphs have a low degree. In contrast, in `RDF` graphs node URIs are used as unique identifiers and hence occur only once in a graph. Moreover, nodes in `RDF` graphs may have a very high degree.

In our approach we condition the computation of the graph kernel to a graph matching problem. We say that two items are similar if they share a similar neighborhood in the `RDF` graph. The rationale behind our approach is that two items are similar if they are related to similar entities. Each item $i \in I$ is then modelled by extracting a graph-based representation from the underlying `RDF` data. Given the `URI` corresponding to i, we perform a breadth-first search (via `SPARQL` queries) from this item up to a limited depth to extract a subgraph which characterizes the information content associated to i. In the approach presented

[1] Following the notation used in the kernel literature, we use $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ to denote the inner product between the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

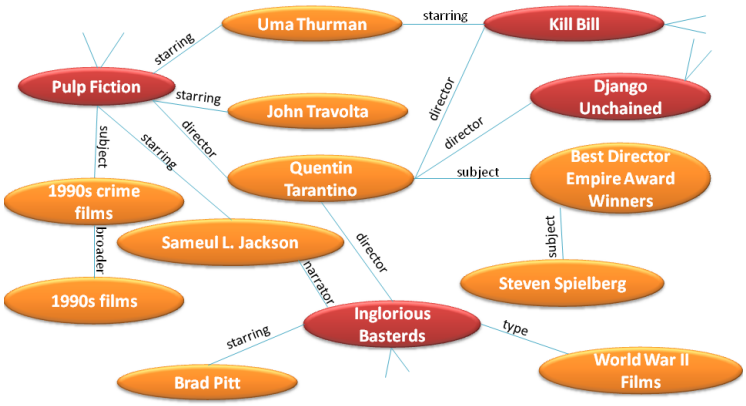

Fig. 1. Fragment of an RDF graph extracted from DBpedia.

in this paper, we consider the underlying RDF graph as undirected because we are mostly interested in the entities and not in the relation type in itself. So, given an RDF entity e, we collect all the RDF triples where it appears as either subject or object via a property p. Collecting all the RDF triples extracted starting from each item we build a new undirected graph $G = (E, P)$, where E represents the set of nodes (entities) and $P \subseteq E \times E$ the set of edges. We name each edge in P as $p_k(e_n, e_m)$ to denote that the property p_k connects the two entities e_n and e_m. This means that either the triple (e_n, p_k, e_m) or the triple (e_m, p_k, e_n) belongs to the original RDF graph. We notice that, by construction of G, we have $I \subset E$. In this work, we do not consider literals nor datatype properties in the construction of the graph.

Fig. 1 shows a sketch of the graph G extracted from DBpedia for the movie domain. The nodes in red belong to I and represent movies. As for the other entities in the figure, it is easy to notice they represent actors, directors, narrators, categories, classes. In the actual graph, there are many more properties that allow us to get rich and detailed knowledge about movies.

We introduce now the notion of **h-hop item neighborhood graph** for G. For a generic item i, its h-hop neighborhood graph $G^h(i) = (E^h(i), P^h(i))$ is the subgraph of G induced by the set of entities $E^h(i)$ that are reachable in **at most** h hops from i according to the shortest path. Fig. 2 shows two possible 2-hop item neighborhood graphs for item i and item j, respectively $G^2(i)$ and $G^2(j)$. We see that, if we consider the shortest path, all the entities are no more than 2 hops distant from i and j, respectively.

Finally, we define $\widehat{E}^h(i) = E^h(i) \setminus E^{h-1}(i)$ as the set of entities **exactly** h hops far from i. In other words, these entities are reachable, based on the shortest path, only after h hops. Analogously, we define $\widehat{P}^h(i) = P^h(i) \setminus P^{h-1}(i)$.

In order to clarify how to build $\widehat{E}^h$ and $\widehat{P}^h$, we show an example using the two item neighborhood graphs in Fig. 2. With reference to items i and j, we have:

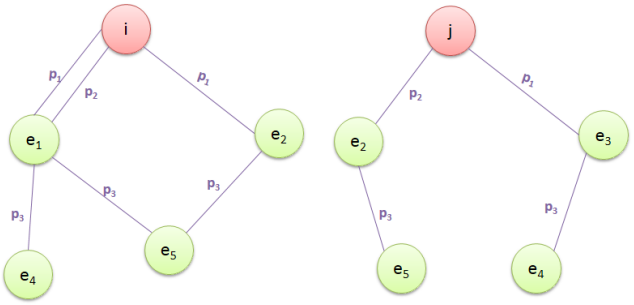

Fig. 2. Two 2-hop item neighborhood graphs.

$$
\begin{aligned}
&E^1(i) = \{e_1, e_2\} && P^1(i) = \{p_1(i,e_1), p_2(i,e_1), p_1(i,e_2)\}\\
&E^2(i) = \{e_1, e_2, e_4, e_5\} && P^2(i) = \{p_1(i,e_1), p_2(i,e_1), p_1(i,e_2), p_3(e_1,e_4),\\
& && \quad p_3(e_1,e_5), p_3(e_2,e_5)\}\\
&\widehat{E}^1(i) = \{e_1, e_2\} && \widehat{P}^1(i) = \{p_1(i,e_1), p_2(i,e_1), p_1(i,e_2)\}\\
&\widehat{E}^2(i) = \{e_4, e_5\} && \widehat{P}^2(i) = \{p_3(e_1,e_4), p_3(e_1,e_5), p_3(e_2,e_5)\}
\end{aligned}
$$

$$
\begin{aligned}
&E^1(j) = \{e_2, e_3\} && P^1(j) = \{p_2(j,e_2), p_1(j,e_3)\}\\
&E^2(j) = \{e_2, e_3, e_4, e_5\} && P^2(j) = \{p_2(j,e_2), p_1(j,e_3), p_3(e_2,e_5), p_3(e_3,e_4)\}\\
&\widehat{E}^1(j) = \{e_2, e_3\} && \widehat{P}^1(j) = \{p_2(j,e_2), p_1(j,e_3)\}\\
&\widehat{E}^2(j) = \{e_4, e_5\} && \widehat{P}^2(j) = \{p_3(e_2,e_5), p_3(e_3,e_4)\}
\end{aligned}
$$

As we will see in Section 2.2, $\widehat{E}^h(i)$ and $\widehat{P}^h(i)$ form the basis for our neighborhood-based graph kernel. We may observe that $\widehat{E}^h(i)$ contains information about which entities are associated to item i at a given distance, while $\widehat{P}^h(i)$ about how they occur.

Finally, we can see that if $G^h(i)$ and $G^h(j)$ share at least one entity, then i and j are at most distant $2 \cdot h$ hops. For instance, for a *2-hop item neighborhood graph* as the ones depicted in Fig. 2 we can compute paths of up to length 4 between i and j.

Summing up, let $I = \{i, j, \ldots\}$ be the set of items we want to recommend, as for example movies or musical artists, we represent them by their h-hop neighborhood graph $G^h(i)$. This graph-based data is exploited by our kernel to build a `LOD`-based recommender system able to suggest relevant resources in I, given T_u.

2.2 Neighborhood-based graph kernel

Based on the notion of *h-hop item neighborhood graph* we define the **h-hop neighborhood-based graph kernel** $k_{G^h}(i,j)$ as:

$$k_{G^h}(i,j) = \langle \phi_{G^h}(i), \phi_{G^h}(j) \rangle \tag{1}$$

where the application of the map function ϕ_{G^h} to the h-hop item neighborhood graph $G^h(i)$ gives us its feature vector representation:

$$\phi_{G^h}(i) = (w_{i,e_1}, w_{i,e_2}, ...w_{i,e_m}, ..., w_{i,e_t})$$

where w weights refer to entities in E. Specifically, w_{i,e_m} represents the weight associated to the entity e_m in $G^h(i)$. Each term is computed using the following formula:

$$w_{i,e_m} = \sum_{l=1}^{h} \alpha_l \cdot c_{\widehat{P}^l(i),e_m}$$

where α_l coefficients are real and non-negative, and:

$$c_{\widehat{P}^l(i),e_m} = |\{p_k(e_n, e_m) \mid p_k(e_n, e_m) \in \widehat{P}^l(i) \wedge e_m \in \widehat{E}^l(i)\}|$$

In particular, $c_{\widehat{P}^l(i),e_m}$ is the number of edges in $\widehat{P}^l(i)$ that involve the node e_m, that is the *occurrence* of the entity e_m in the item neighborhood at distance l. The more the entity e_m appears in paths originated by i, the more it is descriptive of i. Indeed, α_l is a weighting factor depending on the distance l from the item i, whose aim is to up-weight entities closer to the item and to penalize farther entities. It allows us to take into account the *locality* of those entities in the graph neighborhood. The closer an entity e_m to the item i, the stronger its relatedness to it. In other words, α_l can be seen as a decay factor for entities farther from the item i. In Section 3.2 we will show the results of the experimental evaluation for different values of α_l.

To clarify how we compute the weights w in the construction of the ϕ_{G^h} feature vectors we show an example using the two item neighborhood graphs represented in Fig. 2. With respect to item i, we have: $c_{\widehat{P}^1(i),e_1} = 2$, $c_{\widehat{P}^1(i),e_2} = 1$, $c_{\widehat{P}^2(i),e_4} = 1$, $c_{\widehat{P}^2(i),e_5} = 2$. For item j, we have: $c_{\widehat{P}^1(j),e_2} = 1$, $c_{\widehat{P}^1(j),e_3} = 1$, $c_{\widehat{P}^2(i),e_4} = 1$, $c_{\widehat{P}^2(i),e_5} = 1$. All other $c_{\widehat{P}^l(i),e}$ are equal to 0. Once the occurrences $c_{\widehat{P}^l,e}$ are known, the computation of the w scores is straightforward.

Finally, the kernel $k_{G^h}(i,j)$ can be computed by taking the scalar product of the respective feature vectors, $\phi_{G^h}(i)$ and $\phi_{G^h}(j)$. The result of the dot product is essentially the weighted count of the common entities shared by the two item neighborhoods. It is noteworthy that this relatedness measure is based both on *occurrence* and on *locality* of the entities.

In order to uniform different neighborhood sizes, each feature vector is normalized to unit length using the $L2$ norm, hence the dot product corresponds to the cosine similarity measure.

Similarly to [20] and [3], our kernel relies on an explicit computation of the feature vectors. This leads to a sparse vector representation that can speed up the computation with respect to pairwise item computation, allowing us to use fast linear SVM solvers.

3 Experimental Evaluation

In this section we detail the experiments accomplished to evaluate our approach. In particular we are interested in the following two aspects: (i) evaluate the accuracy and novelty of the proposed neighborhood-based graph kernel for recommendations leveraging LOD; (ii) investigate the improvements of the proposed approach with respect to a previous work on LOD-based RSs [4] and a RS based on another graph kernel for RDF [11].

3.1 Experimental Setting

Datasets description. The evaluation has been carried out on the well known MovieLens[2] dataset. The original MovieLens 1M dataset contains 1,000,209 ratings for 3,883 movies by 6,040 users. Starting from this dataset, we first transformed the 1-5 star ratings to binary relevance scores using 4 as threshold. Then, in order to exploit the knowledge encoded in LOD datasets, we mapped items in MovieLens to their corresponding DBpedia URI. In this work, we leveraged DBpedia 3.9, one of the principal knowledge bases in the the LOD cloud. It currently contains about 4 million resources, out of which 3.22 million are classified in a consistent ontology[3]. For a detailed explanation about the mapping methodology please refer to [15]. The item mappings to DBpedia are available at: http://sisinflab.poliba.it/semanticweb/lod/recsys/datasets/.

Then, for each movie in the dataset we extracted its h-hop neighborhood graph with $h = 2$. This choice is driven by a preliminary analysis where we found out that $h = 2$ is a good compromise between computation time and accuracy of the recommendation. During the search, we considered all the RDF object properties belonging to the DBpedia ontology[4], plus three more properties: rdf:type, dcterms:subject and skos:broader. At the end of the data extraction procedure, the resulting graph contains 121,584 entities.

Evaluation Methodology. In our evaluation we focused on accuracy and novelty performances. We measured accuracy by *Precision@N/Recall@N* [8] and *Mean Reciprocal Rank*, and novelty by *Entropy-Based Novelty* [1].
Precision@N is computed as the fraction of *top-N* recommended items in test set that are relevant for the user. *Recall@N* is the fraction of relevant items in the test set which appear in the *top-N* recommendation list.
Mean Reciprocal Rank (*MRR*) is useful for evaluating how early in the list the first relevant recommended item appears. Specifically, *Reciprocal Rank* is the inverse of the position of the first relevant recommendation. The higher, the better. In the computation of these accuracy metrics, we considered only the items that appear in the user test set to populate the *top-N* user recommendation list.
As pointed out by [12], the most accurate recommendations according to the

[2] http://www.grouplens.org/node/73
[3] http://wiki.dbpedia.org/Ontology39
[4] http://mappings.dbpedia.org/server/ontology/classes/Film

standard metrics are sometimes not the recommendations that are most useful to users. In order to assess the utility of a recommender system, it is extremely important to evaluate also its capacity to suggest items that users would not readily discover for themselves, that is its ability to generate novel and unexpected results. The *Entropy-Based Novelty* (*EBN*) expresses the ability of a recommender system to suggest less popular items, i.e. items not known by a wide number of users. In particular, for each user's recommendation list L_u, the novelty is computed as:

$$EBN_u@N = -\sum_{i \in L_u} p_i \cdot \log_2 p_i$$

where:

$$p_i = \frac{|\{u \in U \mid i \text{ is relevant to } u\}|}{|U|}$$

The lower $EBN_u@N$, the better the novelty.

Table 1 shows some statistics about the dataset used for the experiments. In order to assess the performance of the proposed algorithm for different sizes of the training set, we split the `MovieLens` dataset in different chunks of training/test data: 20-80%, 40-60% and 80-20%. Evaluating the performance also with a small training set allows us to understand how immune the system is to shortage of information about users i.e., its immunity to cold-start problem for user.

For each training/test partitioning, the task was to use the training set to train the model and generate the *top-N* recommendation list for each user. After the output was produced, we evaluated the system by the metrics previously defined, according to the data in the test set. We repeated the experiment three times for each condition of the training/test partitioning, with randomly selected samples each time. The results presented in Table 2 are averaged across the three runs.

training/test partitioning	users	items	avg train items per user	avg test items per user
`MovieLens` 20-80%	6038	3148	29.67	116.78
`MovieLens` 40-60%	6036	3148	58.98	87.45
`MovieLens` 80-20%	5992	3148	118.23	29.09

Table 1. Dataset statistics

3.2 Results Discussion

The first part of our tests was carried out to tune the α_l coefficients (as defined in Section 2.2) in order to evaluate the effectiveness of the kernel. For this purpose, we performed several experiments by varying the value of the two coefficients α_1 and α_2. We remember that since our experiments have been performed with $h = 2$, we only have these two coefficients. In particular we considered the ratio $\frac{\alpha_1}{\alpha_2}$. The rationale behind our choice is that the higher $\frac{\alpha_1}{\alpha_2}$, the higher is the

importance given to the entities directly connected to the items (*1-hop* entities). A value of $\frac{\alpha_1}{\alpha_2} < 1$ means that *2-hop* entities are given more importance than *1-hop* entities. In Table 2 we show the results for several values of the α-ratio.
Accuracy. When we have a few ratings in the training set (as for `MovieLens` 20-80%), we notice the best accuracy results are reached for values $\frac{\alpha_1}{\alpha_2} \leq 1$. To us, this is quite surprisingly since it means that, if we want to improve the accuracy of recommended items, the contribution carried by entities at a 2-hop distance must be considered more or equally relevant than the one of the 1-hop distant entities. Our interpretation for this behavior is that when the training set is small, the system has to learn the user model based on a few items in the user profile. Hence, up-weighting the entities at 2-hop distance allows the system to better exploit the `LOD` graph structure by catching implicit relations between items. The situation changes when the training set grows.
Novelty. The best novelty results are achieved when the α-ratio is equal to 1, under all the conditions of partitioning. This means giving the same importance to *1-hop* and *2-hop* entities.

Comparison with other methods. In the second part of our experiments we compared our approach with other existing RSs. Table 3 shows the results. `NK-`α`-ratio` refers to our `N`eighborhood-based `K`ernel approach. `VSM` is the `LOD`-based RS presented in [4]. It relies on a bag-of-resources item representation and the well known `V`ector `S`pace `M`odel. The user model is learned using SVM. Both in [4] and in our approach the C meta-parameter for SVM is chosen via cross-validation. `NB` uses the same item feature representation as [4], but a different learning algorithm, the `N`aive `B`ayes classifier. This is the baseline for our comparison. `WK` refers to the `W`alk-based `K`ernel presented in [11] for dealing with `RDF` graphs. The procedure to learn the user model is the same, we only replaced the kernel. The results show that our approach performs always better than the others, both in accuracy and in novelty.

4 Related Work

Being our approach content-based and leveraging `Linked Open Data`, in this section we will overview RSs based on ontologies and on `LOD`. Part of this section is also reserved to existing literature about graph kernels.
Ontology-based RSs. *Foxtrot* and *Quickstep* are two ontology-based recommender systems presented in [13]. The definition of semantic user profiles allows the system to compute collaborative recommendations. In [2] the authors propose a hybrid recommendation system, where user preferences and item features are described by semantic concepts to obtain users' clusters corresponding to implicit *Communities of Interest*. In [14] the authors introduce the so called *semantically enhanced collaborative filtering* where structured semantic knowledge about items is used in conjunction with user-item ratings to create a combined similarity measure for item comparisons. In all of these works, experimental evaluation demonstrates that the accuracy is improved especially in presence of

α_1/α_2	MRR	P@1	P@5	P@10	R@1	R@5	R@10	EBN@10	EBN@25	EBN@50
Training set 20% – Test set 80%										
0.25	**0.8701**	**0.7665**	**0.7708**	**0.6895**	**0.0259**	**0.1300**	**0.2342**	0.3683	0.9110	1.8428
1	0.8454	0.7267	0.7482	0.6712	0.0242	0.1259	0.2304	**0.2919**	**0.7493**	**1.5027**
2	0.8570	0.7473	0.7591	0.6817	0.0249	0.1267	0.2312	0.4417	1.1223	2.3400
5	0.8536	0.7416	0.7551	0.6772	0.0245	0.1257	0.2311	0.3186	0.8239	1.6596
10	0.8541	0.7420	0.7576	0.6824	0.0243	0.1248	0.2310	0.4659	1.0691	1.9882
20	0.8551	0.7430	0.7574	0.6840	0.0245	0.1243	0.2309	0.3642	0.8930	1.8039
Training set 40% – Test set 60%										
0.25	0.8700	0.7662	0.7792	0.6846	0.0351	0.1775	0.3128	0.6298	1.5448	3.0880
1	0.8576	0.7459	0.7671	0.6767	0.0343	0.1770	0.3101	**0.4484**	**1.1523**	**2.3442**
2	0.8676	0.7633	0.7738	0.6822	0.0347	0.1766	0.3110	0.6920	1.7489	3.572
5	0.8626	0.7542	0.7729	0.6787	0.0342	0.1761	0.3096	0.5196	1.3431	2.7098
10	0.8687	0.7637	0.7809	0.6866	0.0344	0.1775	0.3124	0.6955	1.6652	3.1951
20	**0.8726**	**0.7709**	**0.7833**	**0.6882**	**0.0352**	**0.1776**	**0.3129**	0.5665	1.4109	2.8935
Training set 80% – Test set 20%										
0.25	0.8574	0.7465	0.7323	0.5737	0.1028	0.4705	0.6320	1.0336	2.5164	4.9358
1	0.8533	0.7393	0.7325	0.5756	0.1026	0.4738	0.6349	**0.6584**	**1.7063**	**3.5096**
2	0.8537	0.7389	0.7347	0.5754	0.1028	0.4739	0.6341	1.0251	2.5985	5.1991
5	0.8536	0.7393	0.7361	0.5777	0.1008	0.4740	0.6351	0.7945	2.0451	4.1497
10	0.8607	0.7505	0.7440	0.5796	0.1030	0.4771	0.6358	0.9947	2.4532	4.8567
20	**0.8678**	**0.7625**	**0.7448**	**0.5799**	**0.1059**	**0.4775**	**0.6359**	0.8489	2.1117	4.3242

Table 2. Accuracy and Novelty results for `MovieLens` data – kernel calibration.

alg	MRR	P@1	P@5	P@10	R@1	R@5	R@10	EBN@10	EBN@25	EBN@50
Training set 20% – Test set 80%										
NK-0.25	**0.8701**	**0.7665**	**0.7708**	**0.6895**	**0.0259**	**0.1300**	**0.2342**	0.3683	0.9110	1.8428
NK-1	0.8454	0.7267	0.7482	0.6712	0.0242	0.1259	0.2304	**0.2919**	**0.7493**	**1.5027**
NB	0.7908	0.6577	0.6158	0.6099	0.0222	0.1071	0.2146	0.6817	1.4635	2.5681
VSM	0.8341	0.7101	0.7241	0.6633	0.0233	0.1188	0.2251	0.3543	0.8631	1.6847
WK	0.7948	0.6624	0.6167	0.6063	0.0223	0.1077	0.2143	0.5286	1.2648	2.3743
Training set 40% – Test set 60%										
NK-1	0.8576	0.7459	0.7671	0.6767	0.0343	0.1770	0.3101	**0.4484**	**1.1523**	**2.3442**
NK-20	**0.8726**	**0.7709**	**0.7833**	**0.6882**	**0.0352**	**0.1776**	**0.3129**	0.5665	1.4109	2.8935
NB	0.7844	0.6452	0.6211	0.6158	0.0298	0.1451	0.2890	1.1050	2.4112	4.2484
VSM	0.8528	0.7392	0.7518	0.6722	0.0329	0.1679	0.3066	0.5329	1.3481	2.6926
WK	0.7911	0.6523	0.6185	0.6085	0.0298	0.1449	0.2883	0.7477	1.8834	3.6366
Training set 80% – Test set 20%										
NK-1	0.8533	0.7393	0.7325	0.5756	0.1026	0.4738	0.6349	**0.6584**	**1.7063**	**3.5096**
NK-20	**0.8678**	**0.7625**	**0.7448**	**0.5799**	**0.1059**	**0.4775**	**0.6359**	0.8489	2.1117	4.3242
NB	0.7894	0.6506	0.6214	0.5454	0.0918	0.4174	0.6133	1.698	3.7625	6.5729
VSM	0.8509	0.7344	0.7309	0.5741	0.1001	0.4709	0.6340	0.7628	1.9818	4.042
WK	0.7892	0.6475	0.6161	0.5336	0.0923	0.4154	0.6106	0.9915	2.5958	5.0806

Table 3. Accuracy and Novelty results for `MovieLens` data – comparative approaches.

sparse datasets.

LOD-based RSs. In the last years, great interest has been shown by the scientific community in using `Linked Open Data` for RSs. The authors in [7] were among the first to theoretically propose to use `LOD` as knowledge base for recommender systems. In [18] the authors use `DBpedia` to feed a RS based on matrix-factorization. In [4, 5] a model-based approach and a memory-based one are presented to compute content-based recommendations leveraging `LOD` datasets and `DBpedia` in particular. More recently, two hybrid approaches have been presented: in [15] it is shown how to compute *top-N* recommendations from implicit feedback using linked data sources and in [10] the authors propose an event recommendation system based on linked data and user diversity. In [16] a mobile

RS that uses DBpedia as the core for the recommendation is presented.
Graph kernels. Several graph kernels have been proposed in machine learning. Based on the concept of *Convolution Kernels* [6], many kernel functions work by counting common structures in graphs. While the problem of checking whether two graphs are isomorph is known to be NP-complete, the task of searching for common substructure usually can be done more efficiently. In [20] the authors present a kernel based on the Weisfeiler-Lehman test of graph isomorphism. Basically they compute the number of subtrees shared between two graphs. More recently, some variants of these kernels have been developed in the field of Semantic Web with application to RDF graphs. In [11] graph kernels based on intersection graphs and intersection trees are introduced. Finally, a faster approximation of the Weisfeiler-Lehman graph kernel is presented in [3].

5 Conclusion and Future Work

The high-quality and vast information contained within Linked Open Data datasets, makes LOD the perfect candidate for a new era of knowledge-enabled and content-based recommender systems. In this paper we have presented a content-based recommender system that leverages the knowledge encoded in semantic datasets of the Linked Open Data compass. The innovative aspects of this work are the way we represent items in the knowledge base by their neighborhood graphs, and the usage of a neighborhood-based graph kernel that is able to effectively exploit the local neighborhood of such items. The evaluation shows an improvement in the accuracy and novelty of our system with respect to existing approaches for content-based recommendation. Currently, we are working on other graph kernels which consider different substructures such as partial subtrees.

References

1. A. Bellogín, I. Cantador, and P. Castells. A study of heterogeneity in recommendations for a social music service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 1–8, New York, NY, USA, 2010. ACM.
2. I. Cantador, A. Bellogín, and P. Castells. A multilayer ontology-based hybrid recommendation model. *AI Commun. Special Issue on Rec. Sys.*, 21(2-3):203–210, Apr. 2008.
3. G. K. D. de Vries. A fast approximation of the weisfeiler-lehman graph kernel for rdf data. In *ECML/PKDD (1)*, pages 606–621, 2013.
4. T. Di Noia, R. Mirizzi, V. C. Ostuni, and D. Romito. Exploiting the web of data in model-based recommender systems. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 253–256, New York, NY, USA, 2012. ACM.
5. T. Di Noia, R. Mirizzi, V. C. Ostuni, D. Romito, and M. Zanker. Linked open data to support content-based recommender systems. In *Proceedings of the 8th*

International Conference on Semantic Systems, I-SEMANTICS '12, pages 1–8, New York, NY, USA, 2012. ACM.
6. T. Gärtner, P. A. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *COLT*, pages 129–143, 2003.
7. B. Heitmann and C. Hayes. Using linked data to build open, collaborative recommender systems. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, 2010.
8. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, Jan. 2004.
9. A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 901–912, New York, NY, USA, 2011. ACM.
10. H. Khrouf and R. Troncy. Hybrid event recommendation using linked data and user diversity. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 185–192, New York, NY, USA, 2013. ACM.
11. U. Lösch, S. Bloehdorn, and A. Rettinger. Graph kernels for rdf data. In *Proceedings of the 9th International Conference on The Semantic Web: Research and Applications*, ESWC'12, pages 134–148, Berlin, Heidelberg, 2012. Springer-Verlag.
12. S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.
13. S. E. Middleton, D. D. Roure, and N. R. Shadbolt. Ontology-based recommender systems. *Handbook on Ontologies*, 32(6):779–796, 2009.
14. B. Mobasher, X. Jin, and Y. Zhou. Semantically enhanced collaborative filtering on the web. In B. Berendt, A. Hotho, D. Mladeni, M. Someren, M. Spiliopoulou, and G. Stumme, editors, *Web Mining: From Web to Semantic Web*, volume 3209 of *Lecture Notes in Computer Science*, pages 57–76. Springer Berlin Heidelberg, 2004.
15. V. C. Ostuni, T. Di Noia, E. Di Sciascio, and R. Mirizzi. Top-n recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 85–92, New York, NY, USA, 2013. ACM.
16. V. C. Ostuni, G. Gentile, T. Di Noia, R. Mirizzi, D. Romito, and E. Di Sciascio. Mobile movie recommendations with linked data. In *Human-Computer Interaction & Knowledge Discovery @ CD-ARES'13*, IFIP International Cross Domain Conference and Workshop on Availability, Reliability and Security, CD-ARES 2013. Springer, 2013.
17. M. J. Pazzani and D. Billsus. The adaptive web. chapter Content-based Recommendation Systems, pages 325–341. Springer-Verlag, Berlin, Heidelberg, 2007.
18. L. Peska and P. Vojtas. Using linked open data to improve recommending on e-commerce. In *2nd International Workshop on Semantic Technologies meet Recommender Systems & Big Data (SeRSy 2013)*. CEUR-WS, 2013.
19. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.
20. N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, Nov. 2011.