Movie Recommendation with DBpedia

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

[1] Politecnico di Bari – Via Orabona, 4, 70125 Bari, Italy
{mirizzi,ostuni}@deemail.poliba.it, {t.dinoia,disciacio}@poliba.it
[2] Exprivia S.p.A. – Viale A. Olivetti, 11/A, 70056 Molfetta (BA), Italy
azzurra.ragone@exprivia.it

Abstract. In this paper we present MORE (acronym of ***MORE*** *than* ***MO****vie* ***RE****commendation*), a Facebook application that semantically recommends movies to the user leveraging the knowledge within Linked Data and the information elicited from her profile. MORE exploits the power of social knowledge bases (e.g. DBpedia) to detect semantic similarities among movies. These similarities are computed by a Semantic version of the classical Vector Space Model (sVSM), applied to semantic datasets. Precision and recall experiments prove the validity of our approach for movie recommendation. MORE is freely available as a Facebook application.

1 Introduction

The field of recommender systems, from an Information Retrieval (IR) perspective, is in its maturity stage and many applications are available on the Web that recommend items to the end user based on a combination of content-based, collaborative filtering and knowledge-based approaches [16]. In this paper we present MORE[3]: a **movie recommender system** in the Web of Data. Currently, the system relies on one of the most relevant datasets in the Linked Data [3] cloud: DBpedia [4], and on the semantic-enabled version of the *Internet Movie Database* (IMDB): LinkedMDB [11]. It is developed as a Facebook application and uses also a faceted-browsing approach to metadata navigation and exploration. MORE basically exploits the information coming from Linked Data datasets to compute a semantic similarity between movies and provide a recommendation to the user. Since MORE has been implemented as a Facebook application, in order to avoid the *cold start* problem typical of content-based recommender systems, when the user starts using it, we may retrieve information about the movies she likes by grabbing them from her Facebook profile. We use semantic information contained in the RDF datasets to compute a semantic similarity between movies the user might be interested in.

Main contributions of this paper are: (i) presentation of a Facebook application for movie recommendation exploiting semantic datasets; (ii) a Semantic-based Vector Space Model for recommendation of items in Linked Data datasets; (iii) evaluation and validation of the approach with MovieLens dataset.

[3] http://apps.facebook.com/movie-recommendation/

The remainder of the paper is structured as follows: in Section 2 we illustrate how we exploit semantic information contained in RDF datasets to compute semantic similarities between movies, then in Section 3 we describe the interface of MORE. In Section 4 we show how to compute similarities between movies using a semantic-adaptation of the Vector Space Model (VSM). Section 5 introduces the recommender system we developed for Linked Data data while Section 6 shows the results of our evaluation. In Section 7 we review relevant related work. Conclusion and future work close the paper.

2 Social knowledge bases for similarity detection

By exploiting its SPARQL endpoint[4], it is possible to ask complex queries to DBpedia with high precision in the results. For example, we may retrieve which are the movies where *Al Pacino* and *Robert De Niro* starred together, and discover that *Righteous Kill*[5] and *Heat*[6] are two of these movies. Intuitively, we assume that these movies are related with each other, since they share part of the cast. Via SPARQL queries, we may also find that there are other characteristics shared between the two movies, such as some categories (e.g. *crime films*). Roughly speaking, the more features two movies have in common, the more they are similar. In a few words, a similarity between two movies (or two resources in general) can be detected if in the RDF graph:

- they are directly related: this happens for example if a movie is the sequel of another movie. In DBpedia this state is handled by the properties dbpedia-owl:subsequentWork and dbpedia-owl:previousWork.
- they are the subject of two RDF triples having the same property and the same object, as for example when two movies have the same director. In the movie domain, we take into account about 20 properties, such as dbpedia-owl:starring and dbpedia-owl:director. They have been automatically extracted via SPARQL queries. The property dcterms:subject needs a dedicated discussion, as we will see in the following.
- they are the object of two RDF triples having the same property and the same subject.

Categories and genres. Categories in Wikipedia are used to organize the entire project, and help to give a structure to the whole project by grouping together pages on the same subject. The sub-categorization feature makes it possible to organize categories into tree-like structures to help the navigation of the categories. In DBpedia, the hierarchical structure of the categories is modeled through two distinct properties, dcterms:subject and skos:broader. The former relates a resource (e.g. a movie) to its categories, while the latter is used to relate a category to its parent categories. Hence, the similarity between two

[4] http://dbpedia.org/sparql
[5] http://en.wikipedia.org/wiki/Righteous_Kill
[6] http://en.wikipedia.org/wiki/Heat_(1995_film)

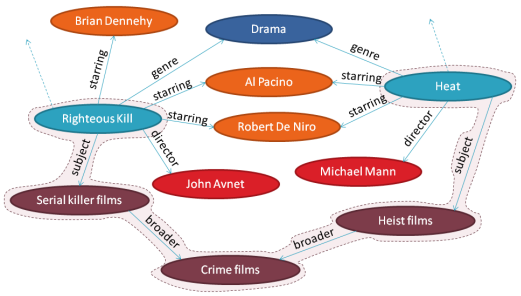

Fig. 1. A sample of an RDF graph related to the movie domain.

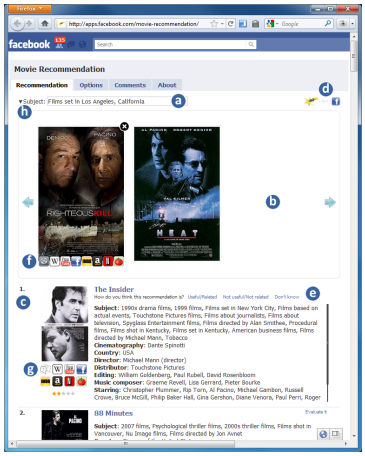

Fig. 2. A screenshot of MORE.

movies can be also discovered in case they have some ancestor categories in common (within the hierarchy). This allows one to catch implicit relations and hidden information, i.e. information that is not directly detectable looking only at the nearest neighbors in the RDF graph. As an example, thanks to the categories, it is possible to infer a relation between *Righteous Kill* and *Heat*, since they both belong (indirectly) to the *Crime films* category, as shown with the highlighted path in Fig. 1, which shows a sample of the RDF graph containing properties and resources coming both from DBpedia and from LinkedMDB/IMDB..

3 MORE: More than Movie Recommendation

In this section we describe MORE, our Facebook application for movie recommendation. A screenshot of the application is depicted in Fig. 2. Although the application exploits semantic datasets, the complex semantic nature of the underlying information is hidden to the end user. She does not interact directly with Semantic Web languages and technologies such as RDF and SPARQL. Despite the choice of the movie domain, we stress that, since our system relies on semantic knowledge bases, it is potentially able to generate recommendations for any areas covered by DBpedia and, more generally, for any dataset in the Linked Data cloud.

After the application is loaded, the user may search for a *movie* by typing some characters in the corresponding text field, as indicated by *(a)* in Fig. 2. The system returns an *auto-complete list* of suggested movies, ranked by popularity in DBpedia. In order to rank the movies in the *auto-complete list*, we adapted the PageRank algorithm to the DBpedia subgraph related to movies. To this aim we consider the property dbpedia-owl:wikiPageWikiLink which corresponds to links between Wikipedia pages. In ranking the results shown in the auto-complete list, we consider also non-topological information by weighting the

results coming from the previous computation with votes on movies from IMDB users.

Once the list has been populated, the user can select one of the suggested movies. Then, the chosen movie is placed in the user's favorite movies area (see *(b)* in Fig. 2) and a **recommendation** of the top-40 movies related to the selected one is presented to the user (see *(c)* in Fig. 2). The relevance rankings for the movies are computed (off-line) as detailed in Section 4. The user can add more movies to her favorite list, just clicking either on its poster or on its title appearing in the recommendation list. Then, the movie is moved into the favorite area and the recommendation list is updated taking into account also the item just added. Another way to add a movie to the favorite list is to exploit the functionalities offered by the Facebook platform and the Graph API[7]. Facebook users can add their favorite movies to their own Facebook profile. In MORE, the user can obtain her preferred Facebook movies by clicking on the icon indicated with *(d)* in Fig. 2. Then, the user can select a movie from the returned list, in order to add it to the favorite area and to obtain the related recommendation. Each of these actions are tracked by the system. In fact, our long run goal is to collect relevant information about user preferences in order to provide a personalized recommendation that exploits both the knowledge bases such as DBpedia or LinkedMDB (**content-based** approach) and the similarities among users (**collaborative-filtering** approach). The user is allowed to set her personal preferences about the properties involved in the recommendation using the sliders in the *Options* tab. In Section 4 we will detail how we automatically compute a default value for the weights associated to each property.

4 Semantic Vector Space Model

In order to compute the similarities between movies, we propose a semantic-adaptation of one of the most popular models in classic information retrieval [1]: the Vector Space Model (VSM) [17]. In VSM non-binary weights are assigned to index terms in queries and in documents (represented as sets of terms), and are used to compute the degree of similarity between each document in the collection and the query. In our approach, we semanticized the classical VSM, usually used for text retrieval, to deal with RDF graphs. In a nutshell, we represent the whole RDF graph as a 3-dimensional tensor where each slice refers to an ontology property. Given a property, each movie is seen as a vector, whose components refer to the *term frequency-inverse document frequency* TF-IDF (or better, in this case, *resource frequency-inverse movie frequency*). For a given slice (i.e. a particular property), the similarity degree between two movies is the correlation between the two vectors, and it is quantified by the cosine of the angle between them. An RDF graph can be viewed as a labeled graph $G = (V, E)$, where V is the set of RDF nodes and E is the set of predicates (or properties) between nodes in V. In our model, an RDF graph is then a 3-dimensional tensor $\mathbf{T}$ where each slice identifies an adjacency matrix for an RDF property (see Fig. 3). All the nodes in

[7] http://developers.facebook.com/docs/reference/api/

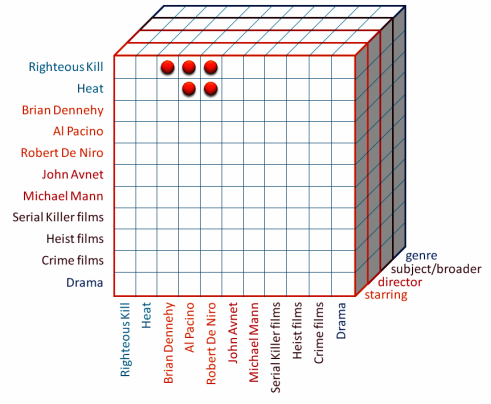

Fig. 3. Tensor representation of the RDF graph of Fig. 1. Only the components on the first slice (i.e. *starring*) are visible.

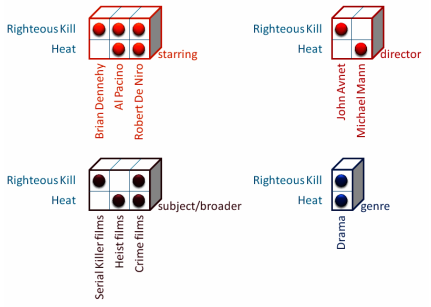

Fig. 4. Slices decomposition.

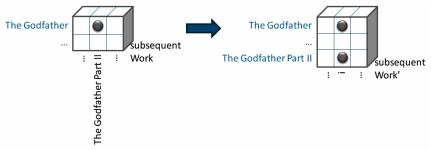

Fig. 5. Property transformation.

V are represented both on the rows and on the columns. A component (i.e. a cell in the tensor) is not *null* if there is a property that relates a subject (on the rows) to an object (on the columns). A few words need to be spent for the properties `dcterms:subject` and `skos:broader`. As also shown in Fig. 1 every movie is related to a category by the property `dcterms:subject` which is in turn related to other categories via `skos:broader` organized in a hierarchical structure. In order to catch such a relation, for each resource we computed the transitive closure of the category it is related to and assign the whole set of computed categories as the value of `dcterms:subject` of the corresponding movie. As an example, going back to the small example depicted in Fig. 1, the set of values assigned to `dcterms:subject` for *Righteous Kill* is {*Serial Killer films*, *Crime Films*}. This can be viewed as an explicit representation of the two following triples:

```
dbpedia:Righteous_Kill dcterms:subject dbpedia:Category:Serial_killer_films
dbpedia:Righteous_Kill dcterms:subject dbpedia:Category:Crime_films
```

Looking at the model, we may observe and remember that: (1) the tensor is very sparse; (2) we consider properties as independent with each other (there is no `rdfs:subPropertyOf` relation); (3) we are interested in discovering the similarities between movies (or in general between resources of the same `rdf:type` and not between any pair of resources). Based on the above observations, we can decompose the tensor slices into smaller matrices. Each matrix of Fig. 4 refers to a specific RDF property, and corresponds to a slice in the tensor. In other words, for each matrix, the rows represent somehow the *domain* of the considered property, while the columns its *range*. For a given property, the components of each

row represent the contribution of a resource (i.e. an actor, a director, etc.) to the corresponding movie. With respect to a selected property p, a movie m is then represented by a vector containing all the terms/nodes related to m via p. As for classical Information Retrieval, the index terms $k_{n,p}$, that is all the nodes n linked to a movie by a specific property p, are assumed to be all mutually independent and are represented as unit vectors of a t-dimensional space, where t is the total number of index terms. Referring to Fig. 4, the index terms for the *starring* property are *Brian Dennehy*, *Al Pacino* and *Robert De Niro*, while $t = 3$ is the number of all the actors that are objects of a triple involving *starring*. The representation of a movie m_i, according to the property p, is a t-dimensional vector given by:

$$\overrightarrow{m_{i,p}} = (w_{1,i,p}, w_{2,i,p}, ..., w_{t,i,p})$$

where $w_{n,i,p}$ is a non-negative and non-binary value representing the weight associated with a term-movie pair $(k_{n,p}, \overrightarrow{m_{i,p}})$. The weights $w_{n,i,p}$ we adopt in our model are TF-IDF weights. More precisely they are computed as:

$$w_{n,i,p} = f_{n,i,p} * \log\left(\frac{M}{a_{n,p}}\right)$$

where $f_{n,i,p}$ represents the TF, i.e. the frequency of the node n, as the object of an `RDF` triple having p as property and the node i as subject (the movie). Actually, this term can be at most 1, since two identical triples can not coexist in an `RDF` graph. Then, in case there is a triple that links a node i to a node n via the property p, the frequency $f_{n,i,p}$ is 1, otherwise $f_{n,i,p} = 0$, and the corresponding weight $w_{n,i,p}$ is set to 0. M is the total number of movies in the collection, and $a_{n,p}$ is the number of movies that are linked to the resource n, by means of the predicate p. As an example, referring to Fig. 4, for the *starring* property, and considering $n = AlPacino$, then $a_{AlPacino,starring}$ is equal to 2, and it represents the number of movies where *Al Pacino* acted. Relying on the model presented above, each movie can be represented as a $t \times P$ matrix (it corresponds to a horizontal slice in Fig. 3), where P is the total number of selected properties. If we consider a projection on a property p, each pair of movies, m_i and m_j, are represented as t-dimensional vectors. As for classical VSM, here we evaluate the degree of similarity of m_i with respect to m_j, as the correlation between the vectors $\overrightarrow{m_i}$ and $\overrightarrow{m_j}$. More precisely we calculate the cosine of the angle between the two vectors as:

$$sim^p(m_i, m_j) = \frac{\overrightarrow{m_{i,p}} \bullet \overrightarrow{m_{j,p}}}{|\overrightarrow{m_{i,p}}| \times |\overrightarrow{m_{j,p}}|} = \frac{\sum_{n=1}^{t} w_{n,i,p} \cdot w_{n,j,p}}{\sqrt{\sum_{n=1}^{t} w_{n,i,p}^2} \cdot \sqrt{\sum_{n=1}^{t} w_{n,j,p}^2}}$$

Such a value is the building block of our content-based recommender system. By means of the computed similarities, it is possible to ask the system questions like *"Which are the most similar movies to movie m_i according to the specific property $\tilde{p}$?"*, and also "*Which are the most similar movies to movie m_i according to the whole knowledge base?*". In the following we will see how to combine such values with a user profile to compute a content-based recommendation.

5 Semantic content-based Recommender System

The method described so far is general enough and it can be applied when the similarity has to be found between resources that appear as subjects or object of RDF triples[8]. Another case is about how to discover a similarity between resources that are directly related by some specific properties. In the considered movie domain, this situation happens for example with the *subsequentWork* property. In our approach we operate a matrix transformation to revert this situation to the one considered so far. The transformation is illustrated in Fig. 5. In order to use the VSM with two resources directly linked, the property p is transformed into the property p' and its domain remains unchanged. The object of the original RDF triple for the new property p' is mapped into a unique index associated to the original object (in Fig. 5, index i is associated with *The Godfather Part II*), and a new RDF triple is created having as subject the original object and as object the index just created. Referring to Fig. 5, *The Godfather Part II* becomes the subject of a new triple, where the predicate is *subsequentWork'* and the object is the index i. Now our semantic VSM can be applied straight.

If we want to provide an answer also to questions like "*Which are the most similar movies to movie m_i according to the user profile?*" we need a step further to represent the user profile. In our setting, we model it based on the knowledge we have about the set of rated movies. In MORE we have information on the movies the user likes. Hence, the profile of the user u is the set:

$$profile(u) = \{m_j \mid u \text{ likes } m_j\}$$

In order to evaluate if a new resource (movie) m_i might be of interest for u — with $m_i \notin profile(u)$ — we compute a similarity $\tilde{r}(u, m_i)$ between m_i and the information encoded in $profile(u)$ via Equation (1).

$$\tilde{r}(u, m_i) = \frac{\sum_{m_j \in profile(u)} \frac{1}{P} \sum_p \alpha_p \cdot sim^p(m_j, m_i)}{|profile(u)|} \tag{1}$$

In Equation (1) we use P to represent the number of properties we selected (see Section 4) and $|profile(u)|$ for the cardinality of the set $profile(u)$. The formula we adopted to compute $\tilde{r}(u, m_i)$ takes into account the similarities between the corresponding properties of the new item m_i and $m_j \in profile(u)$. A weight α_p is assigned to each property representing its worth with respect to the user profile. If $\tilde{r}(u, m_i) \geq 0.5$ then we suggest m_i to u. We want to stress here that, as discussed in the next section, setting a threshold different from 0.5 does not affect the system results.

[8] When the resources to be ranked appear as objects of RDF triples, it is simply a matter of swapping the rows with the columns in the matrices of Fig. 4 and applying again the same algorithm.

	$\alpha_{subject}$	$\alpha_{director}$	α_{writer}	$\alpha_{starring}$	$error$
α^1	0.123	0.039	0.080	0.159	3
α^2	0.024	0.061	0.274	0.433	5
α^3	0.267	0.356	0.188	0.099	3
α^4	0.494	0.428	0.244	0.230	4
α^5	0.082	0.457	0.484	0.051	1

Table 1. Example of values computed after the training.

5.1 Training the system

Although `MORE` allows the user to manually set a value for each α_p, the system automatically computes their default value by training the model via a genetic algorithm. Similarly to an *N-fold cross validation* [16], we split $profile(u)$ in *five* disjoint sets and used alternatively each of them as a *validation set* and the items in the remaining sets as the *training set* of the genetic algorithm. We selected N = 5 because, based on our experimental evaluation, it represents a good trade-off between computational time and accuracy of results. As a matter of fact, every time a new movie is added to $profile(u)$, we re-compute the values of α_p related to u and train again the model for N times. Hence, the higher is N, the more is the time needed to update the result set of the user. During the training step, in order to classify the movies as "*I like*" for u we imposed a threshold of 0.5 for $\tilde{r}(u, m_i)$. It is noteworthy that the threshold can be set arbitrarily since the genetic algorithm computes α_p to fit that value. Hence, if we lower or we raise the threshold the algorithm will compute new values for each α_p according to the new threshold value. After this procedure is completed, we have a set of *five* different values $A_p = \{\alpha_p^1, \ldots, \alpha_p^5\}$ for each α_p. Each value of A_p corresponds to a different round of training. An example of a possible outcome for a small subset of the properties we have in our model is represented in Table 5.1. The last column represents the *misclassification error* computed by the genetic algorithm, i.e., how many resources m_i are not classified as "*I like*" since $\tilde{r}(u, m_i) < 0.5$. Please note that, ideally, the perfect values for α_p would be those returning a misclassification error equal to 0. Indeed, in this step, the movies we consider in our *validation sets* come directly from the user profile. In order to select the best value for each α_p, we considered different options and we tested which one performed better in terms of precision and recall (see Section 6) in the recommendation step. In particular, we evaluated the system performances in the following cases:

$$\alpha_p = \begin{cases} \textbf{min}(\alpha_p^k \in A_p) \\ \textbf{max}\ (\alpha_p^k \in A_p) \\ \textbf{avg}(\alpha_p^k \in A_p) \\ \alpha_p^k \text{ is the } \textbf{median} \text{ of } A_p \\ \alpha_p^k \text{ with the lowest } error \end{cases}$$

The first three options consider an aggregated value computed starting from A_p while the last one consider the tuple with the lower misclassification error. In Figure 6(a) we show how precision and recall of the final recommendation

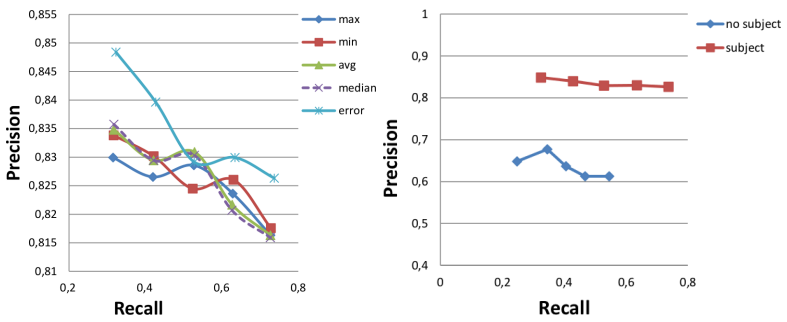

Fig. 6. (a) Precision and recall of the recommendation algorithm with respect to the computation of α_p. (b) Comparison of precision and recall curves with and without `dcterms:subject`.

algorithm vary according to the five cases shown above. We see that the best results are obtained if we consider α_p^k with the lowest misclassification error.

6 Evaluation

In order to evaluate the quality of our algorithm, we performed the evaluation on `MovieLens`, the historical dataset for movie recommender systems. The 100k dataset contains 100,000 ratings from 943 users on 1,682 movies. `MovieLens` datasets are mainly aimed at evaluating collaborative recommender systems in the movie domain. Since our approach is based on a content-based recommendation, in order to use such datasets to test the performances of our algorithms, we linked resources represented in `MovieLens` to `DBpedia` ones. We extracted the value of `rdfs:label` property from all the movies in `DBpedia`, together with the year of production, via `SPARQL` queries. Then, we performed a one-to-one mapping with the movies in `MovieLens` by using the Levenshtein distance and checking the year of production. We found that 78 out of 1,682 (4.64%) movies in `MovieLens` have no correspondence `DBpedia`. After this automatic check we manually double-checked the results and we found that 19 out of 1,604 mappings (1.18%) were not correct and we manually fixed them. Once we had `MovieLens` and `DBpedia` aligned, we tested our pure content-based algorithm by splitting, for each user, the dataset in a *training set* and in a *test set* as provided on the `MovieLens` web-site (80% of the movies rated by the user as belonging to the training set and the remaining 20% as belonging to the *test set*). Before we started our evaluation, we had to align also the user profiles in `MORE` with the ones in `MovieLens`. Indeed, while in more we have only "*I like*" preferences, in `MovieLens` the user u may express a rate on a movie m_j based on a five-valued scale: $r(u, m_j) \in [1, \ldots, 5]$. Hence, following [2] and [15] we build $profile(u)$ as

$$profile(u) = \{m_j \mid r(u, m_j) \in [4, 5]\}$$

In other words, we consider that u *likes* m_j if they rated it with a score greater or equal to 4 and then they are considered as *relevant* to u. The same consideration holds when we evaluate the recommendation algorithm in terms of precision and recall. In recommender systems, precision and recall are defined respectively as: *precision*: fraction of the top-N recommended items that are relevant to u; *recall*: fraction of the relevant items that are recommended to u. In our experiments, since we focus on the test set to find the actual relevant items of the target user, the top-N list we compute only contains items that are in the target user's test set. We varied N in $\{3, 4, 5, 6, 7\}$ and computed the so-called precision@N and recall@N [1]. We did not consider values with N$>$ 7 since in the `MovieLens` dataset we used there are only a few users who rated more than 7 movies as *relevant*. Precision and recall results for `MORE` are shown in Figure 6(a). We also ran our algorithm without taking into account the property `dcterms:subject` in the movie description. The aim of this experiment was to evaluate how important is the ontological information contained in the `DBpedia` categories in the recommendation process. After all, this information can be found only in ontological datasets. In Figure 6(b) we compare precision and recall graphs both when we consider the knowledge carried by `dcterms:subject` and when we do not use it. As we expected, if we do not consider ontological information, the recommendation results get worse drastically.

7 Related Work

`MORE` is intended to be a meeting point between exploratory browsing and content-recommendation in the Semantic Web, exploiting the huge amount of information offered by the Web of Data. Several systems have been proposed in literature that address the problem of movie recommendations, even if there are very few approaches that exploit the `Linked Data` initiative to provide semantic recommendations. In the following we give a brief overview of semantic-based approaches to (movie) recommendation. Szomszor et al. [19] investigate the use of folksonomies to generate tag-clouds that can be used to build better user profiles to enhance the movie recommendation. They use an ontology to integrate both `IMDB` and `Netflix` data. However, they compute similarities among movies taking into account just similarities between movie-tags and keywords in the tag-cloud, without considering other information like actors, directors, writers as we do in `MORE`. *Filmtrust* [9] integrates Semantic web-based social networking into a movie recommender system. Trust has been encoded using the `FOAF` Trust Module and is exploited to provide predictive movie recommendation. It uses a collaborative filtering approach as many other recommender systems, as *MovieLens* [12], *Recommendz* [8] and *Film-Consei* [14]. Our `RDF` graph representation as a three-dimensional tensor has been inspired by [7]. Tous and Delgado [20] use the vector space model to compute similarities between entities for ontology alignment, however with their approach it is possible to handle only a subset of the cases we consider, specifically only the case where resources are directly linked. Eidon et al. [6] represent each concept in an `RDF` graph as a vector con-

taining non-zero weights. However, they take into account only the distance from concepts and the sub-class relation to compute such weights. Effective user interfaces play a crucial role in order to provide a satisfactory user experience during an exploratory search or a content recommendation. Nowadays, there are some initiatives that exploit the Linked Data cloud to provide effective recommendations. One of these is *dbrec* [13], a music content-based recommender system that adopts an algorithm for *Linked Data Semantic Distance*. It uses DBpedia as knowledge base in the Linked Data cloud. The recommendation is link-based, i.e. the "semantics" of relations is not exploited since each relation has the same importance, and it does not take into account the links hierarchy, expressed in DBpedia through the DCTERMS and SKOS vocabulary.

One of the main issues collaborative-filtering recommenders suffer from is the well known *cold-start* problem [18], where no user preference information is known to be exploited for recommendations. In such cases, almost nothing is known about user preferences [10]. Being our system developed as a Facebook application, it is able to automatically extract the favorite movies from the user profile and to provide recommendations also for new users. In [5] the authors propose a hybrid recommender system where user preferences and item features are part of a semantic network. Partially inspired by this work, we offer the capability of inferring new knowledge from the relations defined in the underlying ontology. One of the most complex tasks of their approach is the building of the concepts within the semantic network. Being MORE based on Linked Data and DBpedia, we do not suffer from this problem since it is quite easy to extract, via SPARQL queries, a DBpedia subgraph related to the movie domain.

8 Conclusion and Future Work

The use of Linked Data datasets poses new challenges and issues in the development of next generation systems for recommendation. In this paper we have presented MORE, a Facebook application that works as a recommender system in the movie domain. The background knowledge adopted by MORE comes exclusively from semantic datasets. In particular, in this version of the tool we use DBpedia and LinkedMDB to collect information about movies, actors, directors, etc.. The recommender algorithm relies on a semantic version of the classical Vector Space Model adopted in Information Retrieval. We are willing to better integrate MORE in the Linked Data cloud by publishing our recommendation using the *Recommendation Ontology*[9]. From a methodological perspective, we are collecting information from MORE users to implement also a collaborative-filtering approach to recommendation. This is particularly relevant and challenging since the application is integrated with Facebook.

Acknowledgments. The authors acknowledge partial support of HP IRP 2011. Grant CW267313.

[9] http://purl.org/ontology/rec/core#

References

1. R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search*. Addison-Wesley Professional, 2011.
2. C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *In Proc. of the 15th National Conf. on Artificial Intelligence*, pages 714–720. AAAI Press, 1998.
3. C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst*, 5(3):1–22, 2009.
4. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semant.*, 7:154–165, September 2009.
5. I. Cantador, A. Bellogín, and P. Castells. A multilayer ontology-based hybrid recommendation model. *AI Commun.*, 21:203–210, April 2008.
6. Z. Eidoon, N. Yazdani, and F. Oroumchian. A vector based method of ontology matching. In *Proc. of 3rd Int. Conf. on Semantics, Knowledge and Grid*, pages 378–381, 2007.
7. T. Franz, A. Schultz, S. Sizov, and S. Staab. Triplerank: Ranking semantic web data by tensor decomposition. In *Proc. of 8th ISWC*, pages 213–228, 2009.
8. M. Garden and G. Dudek. Semantic feedback for hybrid recommendations in recommendz. In *IEEE Int. Conf. EEE'05*, pages 754–759, 2005.
9. J. Golbeck and J. Hendler. Filmtrust: Movie recommendations using trust in web-based social networks. In *Proceedings of the IEEE CCNC*, 2006.
10. H. Guo. Soap: Live recommendations through social agents. In *5th DELOS Workshop on Filtering and Collaborative Filtering*.
11. O. Hassanzadeh and M. P. Consens. Linked Movie Data Base. In *Proceedings of the WWW2009 Workshop on Linked Data on the Web (LDOW2009)*, April 2009.
12. J. Herlocker, J. A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *Proceeding on the ACM 2000 Conference on Computer Supported Cooperative Work*, pages 241–250, 2000.
13. A. Passant. dbrec: music recommendations using dbpedia. In *Proc. of 9th Int. Sem. Web Conf.*, ISWC'10, pages 209–224, 2010.
14. P. Perny and J. Zucker. Preference-based search and machine learning for collaborative filtering: the film-consei recommender system. *Information, Interaction, Intelligence*, 1:9–48, 2001.
15. A. Rashid, S. Lam, A. LaPitz, G. Karypis, and J. Riedl. Towards a scalable nn cf algorithm: Exploring effective applications of clustering. In *Advances in Web Mining and Web Usage Analysis*, LNCS, pages 147–166. 2007.
16. F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.
17. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, November 1975.
18. A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 253–260. ACM, 2002.
19. M. Szomszor, C. Cattuto, H. Alani, K. O'Hara, A. Baldassarri, V. Loreto, and V. D. Servedio. Folksonomies, the semantic web, and movie recommendation. In *4th European Semantic Web Conference*, 2007.
20. R. Tous and J. Delgado. A vector space model for semantic similarity calculation and owl ontology alignment. In *DEXA*, pages 307–316, 2006.