Linked Open Data for content-based recommender systems

Roberto Mirizzi, Tommaso Di Noia,
Vito Claudio Ostuni, Azzurra Ragone
Politecnico di Bari – Via Orabona, 4, 70125 Bari, Italy
`{mirizzi,ostuni}@deemail.poliba.it`, `{t.dinoia,a.ragone}@poliba.it`

Abstract

The World Wide Web is moving from a Web of hyper-linked Documents to a Web of linked Data. Thanks to the Semantic Web spread and to the more recent `Linked Open Data` (`LOD`) initiative, a vast amount of `RDF` data have been published in freely accessible datasets. These datasets are connected with each other to form the so called `Linked Open Data` cloud. The aim of `LOD` is to make available rich datasets to be used as the *knowledge background* of complex applications. As of today, there are tons of `RDF` data available in the Web of Data but only few applications really exploit their potential power. In this paper we show how these data can successfully be used to develop a recommender system (RS) that relies exclusively on the information encoded in the Web of Data. We implemented a content-based RS that leverages on the data available within `Linked Open Data` datasets (in particular `DBpedia` and `LinkedMDB`) in order to recommend movies to the end users. We extensively evaluated the approach and validated the effectiveness of the algorithms with precision and recall tests.

Contents

1 Introduction

Since its beginning, the World Wide Web has grown to an increasingly large container of resources and, above all, of information. This is, with no doubt, an excellent point in favor of the Web – you may find whatever information you need – but there is an aspect we necessarily have to take into account: more and more frequently, the user is overwhelmed by information and it might result difficult to find the right path while navigating across Web resources. In other words, we have to face the problem of *information overload*: more information is produced than we can really consume. In order to cope with such a problem, there has been a flourishing of systems that try to facilitate the users during the process of finding the information they are looking for. Among them, we mention a specific family of such systems for information filtering: **recommender systems** [20]. Their ultimate aim is to allow the information to be reached by the interested users. Indeed, recommender systems suggest new items – documents, products to buy, movies to watch, etc. – starting from some initial information about the end user. Currently, state of the art applications for content-based recommendation mainly (but not exclusively) exploit the information coming from different text sources in order to identify keywords, or patterns of keywords able to model both the user profile and the resources to be suggested. By matching keywords within the user profile and within resource descriptions, such systems recommend new resources.

Though the Web was originally conceived to be used by human users, new data-oriented contents have been produced and made available on the Web with the introduction and development of the Semantic Web idea. In particular, more recently there has been a growing interest in the `Linked Open Data` (`LOD`) initiative [5]. The cornerstone of `Linked Open Data` is making available free and open `RDF` datasets linked with each other. The aim is allowing applications to leverage this vast amount of data to enrich their functionalities and to improve the user experience on the Web. Among the datasets available in the `Linked Open Data` cloud, a first-class position is occupied by `DBpedia` [1], the `RDF`-based version of Wikipedia.

In this paper we show how `Linked Open Data` (and in particular `DBpedia`) is mature enough and rich in high quality data to be used as the main information source for a complete application that offers the functionalities of a recommender system. Indeed, although recommender systems leverage on well established technologies and tools, new challenges arise when they exploit the huge amount of interlinked data coming from the Semantic Web. We show how the well-known *Vector Space Model* (*VSM*) may be used when dealing with semantic information and how exploratory browsing features may live together and support the recommendation process.

The main contributions of this paper may be summarized as in the following:

- *Usage of* `LOD` *as background data for Web applications.* We developed an application that relies exclusively on the data coming from `Linked Open Data` social datasets, in particular `DBpedia` and `LinkedMDB` [13] (i.e., the `RDF`-based version of the popular Internet Movie Database – IMDB). These high quality data have proven to be effective to develop a complex Web application.

- *Creation of a LOD-based recommender system in the field of movies.* Even if the general approach we present here is domain independent, we selected this specific domain in order to ease the comparison with other non-LOD systems. In fact, on the "recommender" side, we have the availability of standard test-sets such as MovieLens [15].
- *Development of a LOD-enabled version of the VSM.* The purpose of focusing on the *VSM* is the creation of a baseline for future analysis and comparisons with other techniques applied to semantic datasets. Indeed, this model, although simple, shows its effectiveness in many real-world scenarios.
- *Evaluation.* Our system has been extensively evaluated both by performance analysis and by standard precision and recall measures against the MovieLens dataset.

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 present our recommendation algorithm. Section 4 is devoted to the description of MORE, the LOD-enabled application we developed for movie recommendation. In Section 5 we evaluate our algorithms and show their effectiveness, while in Section 6 we review relevant related work. Conclusion and future work close the paper.

2 Computing similarity in LOD datasets

In order to show the effectiveness of LOD-enabled applications, we developed MORE (***More*** *than* ***Mo****vie* ***Re****commendation*), a movie recommender system. The recommendation is mainly based on the computation of a (semantic) similarity between movies the user might be interested in. Movies are identified by resources described in two LOD datasets: DBpedia and LinkedMDB.

Thanks to its SPARQL endpoint[1], it is possible to ask complex queries to DBpedia with high precision in the results. For example, via the following query:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
SELECT ?movie WHERE {
   ?movie rdf:type dbpedia-owl:Film .
   ?movie dbpedia-owl:starring dbpedia:Al_Pacino .
   ?movie dbpedia-owl:starring dbpedia:Robert_De_Niro .
}
```

[1]http://dbpedia.org/sparql

we may retrieve which are the movies where *Al Pacino* and *Robert De Niro* starred together, and discover that both *Righteous Kill* and *Heat* 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 (see Section 5.1). 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 they 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 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 another relation between *Righteous Kill* and *Heat*, since they both belong (indirectly) to the *Crime films* category, as shown with the highlighted path in Figure 1.

Figure 1 shows a sample of the `RDF` graph containing properties and resources coming both from `DBpedia` and from `LinkedMDB/IMDB`. In this work we consider only *object properties*, i.e., properties whose object is a URI. We agree that including also *datatype properties*, i.e., properties whose object is a literal, might improve the recommendation results, however in this work our focus was just on resources. Nevertheless, we point out that our approach works also with literal, e.g., if two movies have the same release year, and this information exists in the considered knowledge bases, then the same ideas still apply.

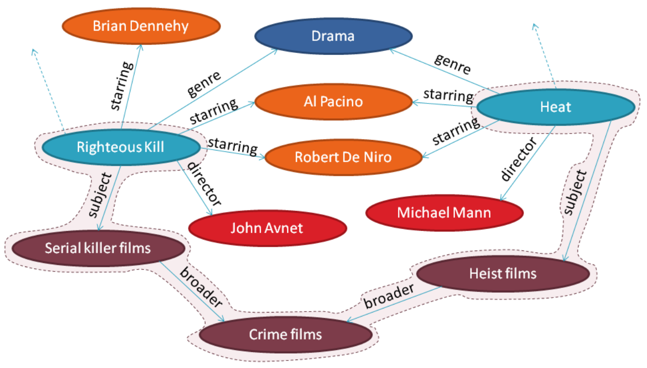


Figure 1: A sample of an `RDF` graph related to the movie domain.

2.1 VSM for LOD

In order to compute the similarities between movies, we adapted to a `LOD`-based setting one of the most popular models in classic information retrieval [2]: the Vector Space Model (VSM) [22]. We want to stress here that, among the various algorithms to compute similarities, such as *Support Vector Machines* and *Nearest Neighbor* approaches [2] just to cite a few, we selected a pure VSM in order to have a good baseline for further developments and comparisons.

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 Figure 2(a)). All the nodes in 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 Figure 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. Note that the formal semantics of `skos:broader` does not states this relation is transitive. If one wants to model a transitive relation, `skos:broaderTransitive` needs to be used. Nevertheless, our experimental evaluation (see Section 5.2) shows for our recommendation application it is useful to consider `skos:broader` as *one-step transitive*. We will explain this notion with the aid of a simple example. Suppose to have the following `RDF` statements:

```
dbpedia:Righteous_Kill
    dcterms:subject dbpedia:Category:Serial_killer_films .
dbpedia:Category:Serial_killer_films
    skos:broader dbpedia:Category:Crime_films .
dbpedia:Category:Crime_films
    skos:broader dbpedia:Category:Films_by_genre .
```

Starting from `dbpedia:Category:Serial_killer_films` we have that `dbpedia:Category:Crime_films` is at a distance of one step, while `dbpedia:Category:Films_by_genre` is at a distance of two steps with respect to `skos:broader`. If we considered `skos:broader` as transitive, then we would have the following `RDF` statements inferred by the previous ones:

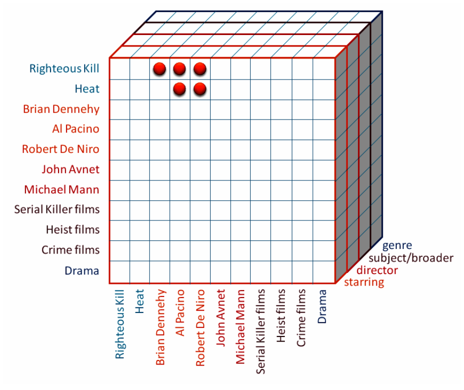
```
dbpedia:Righteous_Kill
    dcterms:subject dbpedia:Category:Crime_films .
dbpedia:Righteous_Kill
    dcterms:subject dbpedia:Category:Films_by_genre .
```

By considering a *one-step transitivity* we have that only:

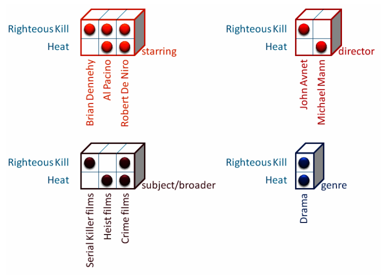
```
dbpedia:Righteous_Kill
    dcterms:subject dbpedia:Category:Crime_films .
```

is inferred by the original statements.

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 each pair of resources). Based on the above observations, we can decompose the tensor slices into smaller matrices where each matrix refers to a specific `RDF` property. 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 Figure 2(b), 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:



(a)



(b)

Figure 2: (a)Tensor representation of the `RDF` graph of Figure 1. Only the components on the first slice (i.e. *starring*) are visible. (b) Slices decomposition.

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

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 Figure 2(b), 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 Figure 2(a)), 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. 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:

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

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?"*.

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. 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 Figure 2(b) and applying again the same algorithm. Lastly, when two resources are directly related by some specific properties (as the case of the property `dbpedia:subsequentWork`), we simply operate a matrix transformation to handle this case the same way as done so far.

In the following we will see how to combine such similarity values with a user profile to compute a content-based recommendation.

3 Semantic content-based Recommender System

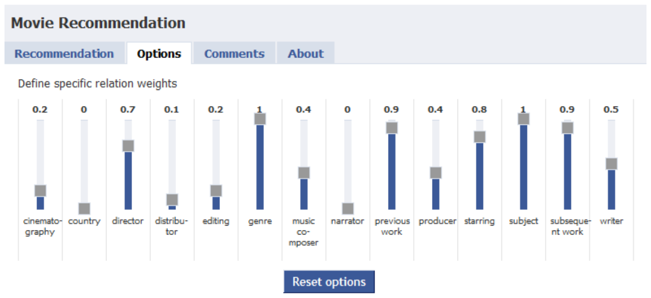
If we want to provide an answer also to questions like *"Which are the most similar movies to movie m_i according to the user profile?"* we need a step further and represent the user profile. In our setting, we model it based only on the knowledge we have about the set of rated movies. In `MORE` we only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

$$profile(u) = \{m_j \;\mid\; u \text{ likes } m_j\} \tag{1}$$

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 the following formula:

$$\tilde{r}(u, m_i) = \frac{\displaystyle\sum_{m_j \in profile(u)} \frac{1}{P} \sum_{p} \alpha_p \cdot sim^p(m_j, m_i)}{|profile(u)|} \tag{2}$$

In Equation (2) we use P to represent the number of properties we selected from the datasets we consider (see Section 2.1) 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.



Define specific relation weights

cinemato-graphy	country	director	distribu-tor	editing	genre	music co-mposer	narrator	previous work	producer	starring	subject	subseque-nt work	writer
0.2	0	0.7	0.1	0.2	1	0.4	0	0.9	0.4	0.8	1	0.9	0.5

Figure 3: Personalization of the user preferences.

3.1 Training the system

Although the α_p weights can be manually set by the user via the Web interface (as shown in Figure 3), the system automatically computes their default value by training the model via a genetic algorithm. We describe also another approach to automatic extraction of default weights, based on `Amazon`. In the evaluation section (cf. Section 5.2 and Figure 6) we will compare the results obtained by these approaches.

3.1.1 Training via a Genetic Algorithm

Genetic Algorithms (GA) are adaptive algorithms that identify a population of solutions by following the paradigm of evolution and genetics [25]. Extensive research on them has proven their effectiveness in solving a large range of optimization problems, including feature selection and weighting tasks. In GAs a fitness function is used to evaluated individuals, and the success of the reproduction of the population varies with the fitness function. In particular, a population of chromosomes, that encode possible solutions, evolves towards better solutions. Referring to our case, the chromosomes are combinations of the α_p coefficients (the genes) and a candidate solution is said to be better than another one if its precision is greater. In other words, our fitness function *maximizes the precision on the validation set*. The evolutionary process starts from a population of α_p coefficients that are generated randomly. Then, in each generation, the fitness function evaluates every possible solution and then multiple solutions are stochastically chosen from the current population, recombined and randomly mutated to obtain a new population. This population is used in the subsequent iteration. In our tests, we varied the number of iterations to decide when the algorithm had to terminate. We observed the quality of the recommendations – measured in terms of Precision and Recall on the test set – does not improve significantly if we consider a number of iterations greater than 100 and a population size greater than 300, while the computational cost increases drastically. For this reason, we set the maximum number of iterations equal to

100 and the population size equal to 300 individuals.

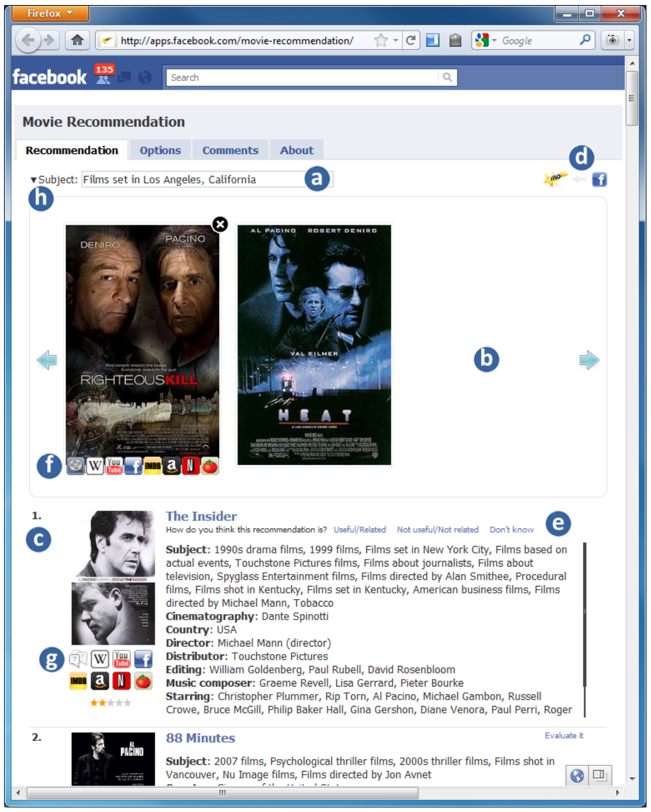
During the training step, in order to classify the movies as "*I like*" for user u we imposed a threshold of 0.5 for $\tilde{r}(u, m_i)$. It is noteworthy that the threshold can be set arbitrarily since the genetic algorithm computes α_p to fit that value. Hence, if we lower or we raise the threshold, the algorithm will compute new values for each α_p according to the new threshold value.

3.1.2 Training via Amazon

The Genetic Algorithm above described computes the optimum values for the α_p coefficients, according to a specific user profile $profile(u)$. However, in case such profile is not available – this happens for example when a new user starts using the application – we use default weights computed via a statistical analysis on `Amazon`'s collaborative recommender system. More in detail, we collected a set of 1000 randomly selected movies from `Amazon` and we checked *why* users that has bought a movie m_i also bought movie m_j, analyzing the first items in the recommendation list. For example, for the movie *Righteous Kill*, the first movie suggested by `Amazon` is *Heat*. Then, looking into our semantic graph, we checked what these two movies have in common. In particular, since these movies share part of the cast, the genre and some categories (cf. Figure 1), we assign an initial value equal to 1 to $\alpha_{starring}$, α_{genre} and $\alpha_{subject/broader}$. The main idea behind this assignment is that a user likes a movie m_i, given another movie m_j, because the two movies are similar according to some properties $p_1, \ldots, p_P$. More precisely, each time there is a path that relates two movies with each other via some property p, a counter associated to p is incremented. We iterate the process on the training movie set, and then we normalize the counters to the highest one, to finally obtain the coefficients α_p in the range $[0, 1]$.

4 MORE: More than Movie Recommendation

In this section we detail how our Web application `MORE` (***Mo**re than **Mo**vie* ***Re**commendation*) works. It has been developed as a `Facebook` application and a screenshot of the application is depicted in Figure 4. Although the application exploits semantic datasets, the complex semantic nature of the underlying information is hidden to the end users. They do not interact directly with Semantic Web languages and technologies such as `RDF` and `SPARQL` and they do not see URIs. The main goals we had in mind during the development of `MORE` were to build an application (i) intuitive and usable for end-users; (ii) rich in information taken from different sources (i.e., we wanted to create an original mash-up); (iii) fast in providing results; (iv) fully integrated with Web 2.0 social applications as `Facebook`. 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 on the `Linked Open Data` cloud. `MORE` has been designed to



▾Subject: Films set in Los Angeles, California

1. **The Insider**

How do you think this recommendation is? Useful/Related Not useful/Not related Don't know

Subject: 1990s drama films, 1999 films, Films set in New York City, Films based on actual events, Touchstone Pictures films, Films about journalists, Films about television, Spyglass Entertainment films, Films directed by Alan Smithee, Procedural films, Films shot in Kentucky, Films set in Kentucky, American business films, Films directed by Michael Mann, Tobacco
Cinematography: Dante Spinotti
Country: USA
Director: Michael Mann (director)
Distributor: Touchstone Pictures
Editing: William Goldenberg, Paul Rubell, David Rosenbloom
Music composer: Graeme Revell, Lisa Gerrard, Pieter Bourke
Starring: Christopher Plummer, Rip Torn, Al Pacino, Michael Gambon, Russell Crowe, Bruce McGill, Philip Baker Hall, Gina Gershon, Diane Venora, Paul Perri, Roger

2. **88 Minutes** Evaluate it

Subject: 2007 films, Psychological thriller films, 2000s thriller films, Films shot in Vancouver, Nu Image films, Films directed by Jon Avnet

Figure 4: A screenshot of MORE.

be multilingual exploiting the multilingual nature of the DBpedia dataset. The language is automatically determined according to the user language set on her Facebook profile.

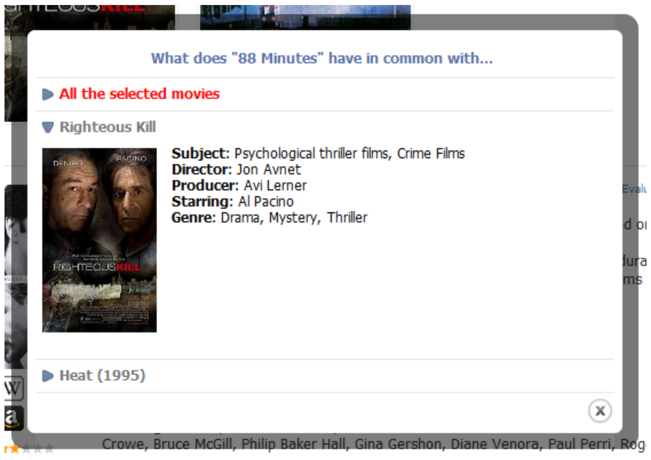
After the application is loaded, the user may search for a *movie* by typing some characters in the corresponding text field, as indicated by *(a)* in Figure 4. The system returns an *auto-complete list* of suggested movies, ranked by popularity in DBpedia. In order to rank the movies in the *auto-complete list*, we adapted the PageRank algorithm to the DBpedia subgraph related to movies. To this aim we consider the property dbpedia-owl:wikiPageWikiLink which represents 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. The list is also sortable by predicted liking, leveraging the user profile as defined in Equation 1. Since this sorting depends on the specific user, it applies only if the systems has enough information about the user profile.

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 Figure 4) and a **recommendation** of the top-40 movies related to the selected one is presented to the user (see *(c)* in Figure 4). The relevance rankings for the movies are computed (off-line) as detailed in Section 2.1. The user can add more movies to his/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 items 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[2]. Facebook users can add favorite movies to their own Facebook profile. In MORE the user can obtain his/her preferred Facebook movies by clicking on the icon indicated with *(d)* in Figure 4. In this case the system tries to map the title of a movie on Facebook to the corresponding title on DBpedia. In particular, if an exact match is found, the unique result is presented to the user. Otherwise, the title is tokenized and a list of potential matches is presented to the user ordered by relevance. Then, the user can select one or more movies from the returned list, in order to add them 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).

4.1 Browsing and Explanation

The interlinked nature of LOD datasets may be very useful to perform exploratory browsing [30] tasks. In MORE we exploit such features in two different ways. On the one hand we allow the user to explore the information related to the results

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



What does "88 Minutes" have in common with...

All the selected movies

Righteous Kill

Subject: Psychological thriller films, Crime Films
Director: Jon Avnet
Producer: Avi Lerner
Starring: Al Pacino
Genre: Drama, Mystery, Thriller

Heat (1995)

Crowe, Bruce McGill, Philip Baker Hall, Gina Gershon, Diane Venora, Paul Perri, Rog

Figure 5: The explanation for the recommended movie.

returned by the recommendation algorithm. On the other hand, we are able to provide an explanation of the returned results to the users. It is noteworthy that both features may support the recommender functionalities. Indeed, by exploring the knowledge space starting from a recommended movie, the users may find a new interesting movie which was not recommended directly by the recommender algorithm.

If the user moves the mouse over the poster in the favourite movies area, a list of icons fades in (see *(f)* in Figure 4). Clicking on the first icon on the left (i.e., the one representing a reel), it is possible to obtain **information** about the selected movie, e.g., *starring*, *director*, *subject*, etc.. Each of these facets corresponds to a `DBpedia` property associated to the movie that can be used to further explore the knowledge associated to the selected movie. The other icons allow one to retrieve information related to the selected movie from `Wikipedia`, `YouTube`, `Facebook`, `Amazon`, `IMDB`, `RottenTomatoes` and `Netflix`. For each movie that appears in the recommendation list, in addition to the just mentioned icons, an icon for **explanation** is also available (see *(g)* in Figure 4). Clicking on it, the user may discover which features the movie shares with the other movies in the favorite area, as shown in Figure 5. In particular, the common values are displayed for each facet (i.e., *subject*, *starring*, etc.). It is well known that adding explanation features to a recommender system is a good choice to increase users' trust [20].

5 Evaluation

In order to assess the performance and the quality of our recommendation algorithm, we analyzed the amount of time required to compute the rankings for the whole dataset and we conducted several precision and recall experiments to evaluate the proposed recommendations.

5.1 Dataset and performance analysis

In the current `DBpedia` release (3.7) there are 60,194 movies. 3,524,733 triples have a movie as subject, almost two-thirds of them (2,288,968) link a movie to a resource (e.g., an *actor* URI, a *director* URI, and not a literal). More precisely, there are 575 distinct properties of such type whose domain is a movie. In our analysis we considered `dcterms:subject`, `skos:broader` and *object properties* (i.e., properties whose type is `owl:ObjectProperty`) while we discarded both the *datatype properties* (i.e., `owl:DatatypeProperty`) and the properties that are not mapped in the `DBpedia` ontology[3]. In particular, we consider `dcterms:subject` and `skos:broader` very relevant for our approach since they encode most of the ontological knowledge in the `DBpedia` dataset. This will be more evident in Section 5.2.

Even if in this work our target considered just resources, we stress that our approach (cf. Section 2.1) works also with literals. Moreover, in `DBpedia` we use only the properties belonging to the DBpedia-Ontology, because they allow users to deal with high-quality, clean and well-structured data. Summing up the most relevant characteristics of the movie subgraph extracted from `DBpedia` and `LinkedMDB`, there are 181,914 triples involving 53,840 distinct *actors*, 54,504 triples referring to 18,149 different *directors*, 68,393 triples concerning 29,352 distinct *writers*.

After the movie subgraph has been extracted, we have measured the runtime performance of the ranking process executed by our algorithm. The program is written in Java and makes extensive use of multi-threading, with 150 concurrent threads. The computation time for the recommendation of the whole extracted dataset lasted 24 minutes and 13 seconds on a dedicated server machine with 4 Xeon quad-core 2.93GHz processors and 32GB RAM. Being an extension of the classical Vector Space Model, our approach has the same time complexity: it is linear in the number of documents (i.e., movies) in the collection. Moreover, several optimizations based on heuristics can be applied straight for speeding up the computation [16].

5.2 Recommender System evaluation

In order to evaluate the quality of our algorithm, we performed the evaluation on `MovieLens` [15], the historical dataset for movie recommender systems. The 100k dataset contains 100,000 ratings from 943 users on 1682 movies. Each user has rated at least 20 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

[3]`http://wiki.dbpedia.org/Downloads37#ontologyinfoboxproperties`

MovieLens to DBpedia ones[4]. We extracted the value of rdfs:label property from all the movies in DBpedia, together with the year of production, by using the following SPARQL query:

```
SELECT DISTINCT ?movie ?label ?year WHERE {
  ?movie rdf:type dbpedia-owl:Film.
  ?movie rdfs:label ?label.
  ?movie dcterms:subject ?cat .
  ?cat rdfs:label ?year .
  FILTER langMatches(lang(?label), "EN") .
  FILTER regex(?year, "^[0-9]{4} film", "i")
}
ORDER BY ?label
```

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 1682 (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 1604 mappings (1.18%) were not correct and we manually fixed them. A dump of the mapping is available at the address: http://sisinflab.poliba.it/mapping-movielens-dbpedia.zip.

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: $r(u, m_j) \in [1, \ldots, 5]$, where a vote of 1 indicates an *Awful* movie, while a vote of 5 classifies a movie as *Must See*. Hence, following [3] and [19], we build $profile(u)$ as:

$$profile(u) = \{m_j \ \mid \ r(u, m_j) \in [4, 5]\} \tag{3}$$

In other words, we consider that u *likes* m_j if user u rated movie m_j with a score greater or equal to 4 and then m_j is considered *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. The 100k MovieLens test set contains exactly 10 rates per user. For this reason, we varied N in the interval $[1, \ldots, 10]$ and computed the Precision@N and Recall@N [2]. More precisely, our experiments of Precision and Recall aim to evaluate three different aspects: (1) comparing

[4]It was not necessary to map also movies in LinkedMDB since they are linked to DBpedia via owl:sameAs statements.

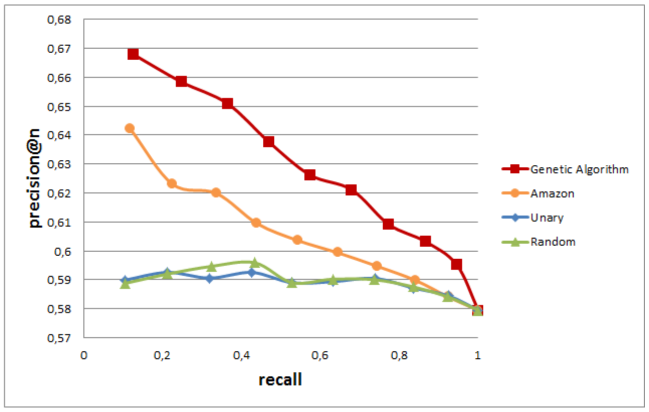


Figure 6: Precision and Recall curves obtained with different algorithms for computing the α_p coefficients.

different algorithms used for the selection of the α_p coefficients (Figure 6), (2) analyzing how choosing specific sets of properties within `DBpedia` affects the quality of results (Figure 7), (3) validating our approach by comparison with existing state-of-the-art recommender systems (Figure 8).

Figure 6 shows the Precision and Recall curves obtained considering different approaches for the computation of the coefficients α_p. In particular, for the curve identified by the square markers, the weights α_p computed via the genetic algorithm are used, as detailed in Section 3.1.1. The curve with round markers refers to the coefficients calculated via statistical analysis on `Amazon` recommender system, as described in Section 3.1.2. Then, the last two curves show the results of Precision and Recall when the coefficients are all set equal to 1 and when they are randomly selected, respectively. In the last case, we ran the random selection 10,000 times (about 10 times the number of users in the `MovieLens` dataset) and then we computed the mean values. For an infinite number of trials, the curve obtained with random selection should be identical to the curve where the coefficients are all equals. This is the reason why the two curves in the figure are very similar. It is evident from Figure 6 that the genetic algorithm allows the system to achieve the best results in term of Precision. For this reason, in the following evaluation we rely on the α_p coefficients computed by this algorithm.

The second part of our evaluation aims to highlight the importance of the ontological information contained within `Linked Open Data` and in particular `DBpedia`. To this purpose, we computed the recommendation lists for users in the test set (via Equation 2), considering specific group of properties. The goal is to understand how these properties affect the quality of the recommendation. The results of Precision and Recall are shown in Figure 7. The curve with square markers considers all the available *object properties* of `DBpedia` and LinkedMDB. Moreover, also the categories not directly related to a movie are considered – i.e., considering skos:broader as *one-step transitive* (see the highlighted path in Figure 1). The best results are achieved in this case. The curve with diamond markers considers the same properties, but the categories not directly related to a movie are excluded (i.e., the skos:broader property is considered as *non-transitive*). It is evident that the result gets worse. Hence, we may say that the information contained in the categories one step far from a movie is very important for the quality of the recommendation. We stress that such type of information can be found only in ontological datasets as the ones belonging to Linked Open Data cloud. The curve with dash markers considers all the properties within the semantic repository, and also the categories one step farther (i.e., the categories of the categories of the categories of movies) – skos:broader is considered as *two-steps transitive*. Also in this case the results worsen: moving away from the movie nodes in the graph causes the inclusion of noise in the information. In fact, the further the categories are, the more general they are. In other words, above a distance of two steps from a movie, the noise starts to be more relevant than the actual information conveyed by these categories. The curve with circle markers considers for the recommendation only dcterms:subject and skos:broader *one-step transitive*, i.e., the categories of a movie and the categories of the categories of a movie, without taking into account the other properties extracted from the DBpedia-Ontology and from LinkedMDB. The results evidence categories by themselves allow the system to obtain "not that bad" Precision, but not as good as the first curve. This means that the ontological information, although very informative, is not sufficient to provide good recommendations. Finally, the last curve, represented with triangle markers, does not consider any categories at all, while all the other properties are included. The worst results are obtained in this very last case: the data considered in this case are comparable to the ones usually available to traditional content-based recommender system, where only textual information is available, since they lack of any ontological representation of the information.

In the last part of the evaluation of our system, we focus on the comparison with existing approaches in the literature to content-based recommendation. The curve with square markers refers to the algorithm used in our system MORE, where the α_p coefficients are obtained via a genetic algorithm as detailed in 3.1.1 and all the object properties within the semantic graph are considered. In particular, skos:broader is considered as *one-step* transitive. All the other curves refer to related work about content-based recommender systems. The curve indicated with diamond markers refers to a work by Wartena et al. [29]. The authors augment the MovieLens dataset with IMDB textual description about movies. Then, keywords are extracted from text and movies are described by bags of words. The main drawback of this approach is that the informative sources considered are not structured. This is the reason why all the other approaches we compare in the following, being based on structured data, obtain better values of Precision/Recall. The curve represented by triangle markers refers to the work by Debnath et al. [7]. Here, the authors leverage on structured data coming from IMDB to produce a content-based recommenda-

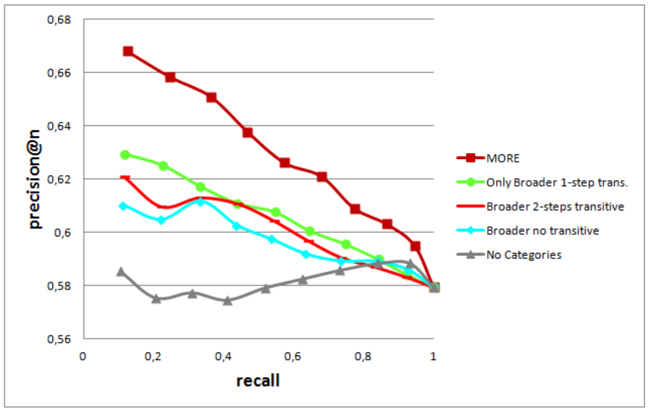


Figure 7: Precision and Recall curves obtained considering different sets of DBpedia properties.

tion. They estimate the values of the weights for the properties involved in the recommendation by a set of linear regression equations obtained from a social network graph. The results are similar to the ones we obtain if we do not consider the information encoded within the DBpedia categories (cf. the curve with triangle markers in Figure 7). Thanks to the ontological information within the Linked Open Data datasets we exploit, we are able to improve the quality of a standard content-based system based on structured data. The curve identified by circle markers refers to CinemaScreen [21], a popular system that combines both collaborative and content-based filtering to recommend new items. From the content-based side, the recommendation leverages only standard features (such as genre, actors and directors). Our algorithm shows better Precision for increasing values of Recall.

6 Related Work

At the best of our knowledge, together with *dbrec* [17], our system is one of the very first initiatives that leverage Linked Open Data to build recommender systems. A lot of systems have been proposed in literature that address the problem of (movie) recommendations, but there are very few approaches that exploit the data within LOD to provide recommendations. In the following we give a brief overview of semantic-based approaches to (movie) recommendation. Szomszor et al. [27] 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* [11] inte-

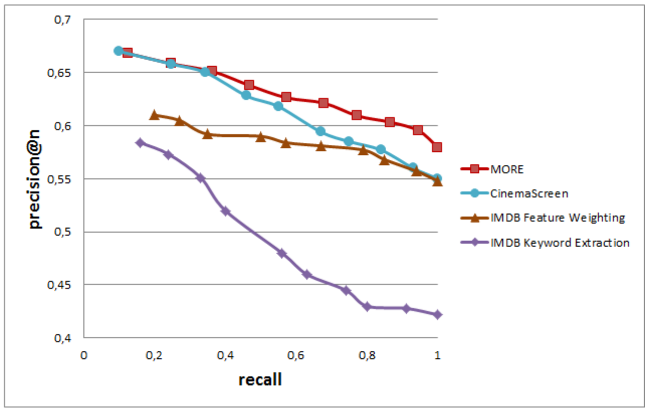


Figure 8: Comparison of `MORE` with content-based RSs.

grates 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* [15], *Recommendz* [10] and *Film-Consei* [18]. Our `RDF` graph representation as a three-dimensional tensor has been inspired by [9]. Tous and Delgado [28] 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. [8] represent each concept in an `RDF` graph as a vector containing non-zero weights. However, they take into account only the distance from concepts and the sub-class relation to compute such weights.

dbrec [17] is a music content-based recommender system leveraging the `DBpedia` dataset. The algorithm used for the recommendation is the *Linked Data Semantic Distance*. The approach 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. However, the properties within these vocabularies have proven to be fundamental in our evaluation (cf. Figure 7). A direct comparison between *dbrec* and our algorithm was not possible because the former is a music RS, while at the present moment our focus was on movies. Being based on `LOD`, in the future we will check the validity of our algorithm also with other domains, such as that of music. Heitmann and Hayes [14] propose to leverage `Linked Open Data` to build open recommender systems. The purpose is to mitigate the new-user, new-item and sparsity problems of collaborative recommender systems. As for [17], the domain is the music. Differently from our approach, they do not exploit any ontological information of `DBpedia`. Moreover, being based on collaborative-filtering, their aim is to fill a user-item matrix with binary values, while we define P vector spaces – P is the number of the properties considered in the recommendation (cf. Section 2.1) – to compute the similarity between resources in order to build a content-based recommender. In [24] the authors propose to use `Linked Open Data` for educational purposes. The presented system is a *Resource List Management System* built on `LOD` principles. However, the recommender part is just a vision.

The explanation for a recommended item is crucial for the acceptance of a recommender system [15]. Content-based recommender systems usually provide justifications for recommendation through *keywords*. In collaborative-filering recommender systems the explanation is based on the *nearest neighbors* [4]. Anyway, both these styles of justification do not motivate conveniently the proposed recommendations, since they are based only on data coming from either keywords or ratings. As for *MoviExplain* [26], `MORE` is able to provide explanations about the recommended movies. Their system relies on voting, interpreting a rating of a movie by an user as a vote to the features of that movie. Nevertheless, their explanations are based only on genre, directors and actors, while we exploit all the information contained within `DBpedia` and `IMDB/LinkedMDB` to identify fine-grained user profiles and provide more detailed explanations. In [26], the content-based part of the recommender system relies on a limited set of features, while as shown in our experimental evaluation (cf. Section 5.2), the information contained in the `DBpedia` categories is critical for the performance of the system.

One of the main issues collaborative-filtering RSs suffer from is the well known *cold-start* problem [23], where no user preference information is known to be exploited for recommendations. In such cases, almost nothing is known about user preferences [12]. 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 [6] 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 Open Data` and `DBpedia`, we do not suffer this problem since it is quite easy to extract, via `SPARQL` queries, a `DBpedia` subgraph related to the movie domain.

7 Conclusion and Future Work

The usage of `Linked Open Data` datasets poses new challenges and issues in the development of next generation recommender system and, more generally, complex Web applications. In this paper we have presented a content-based recommender system that leverages the knowledge encoded in semantic datasets of the `Linked Open Data` project. In particular, since we focused on the movie domain, we exploited the data within `DBpedia` and `LinkedMDB` to collect information about movies, actors, directors, genres, categories, etc.. The innovative aspect of the research is the usage of `Linked Open Data` as the only background knowledge of a movie recommender system, and the adaptation of a popular ranking measure in Information Retrieval as the Vector Space Model to semantic graphs. A Web application for movie recommendation has been built leveraging exclusively the data within `LOD` datasets. The results are supported by an extensive evaluation and comparison with existing state-of-the-art approaches.

Currently, we are working on testing/integrating other approaches for the recommendation (e.g., *Support Vector Machine* or *Nearest Neighbors*) applied to `Linked Open Data`. We will expand the collection of the `LOD` datasets used in the recommendation, and we will extend the whole approach to other domains, to eventually propose a `LOD`-based cross-domain recommendation. Moreover we want to combine a `LOD`-based recommendation with a collaborative-filtering approach to tackle issues typical of CF recommender systems, such as the cold-start problem.

Acknowledgments

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

References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: a nucleus for a web of open data. In *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search.* Addison-Wesley Professional, 2011.

[3] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720. AAAI Press, 1998.

[4] M. Bilgic. Explaining recommendations: Satisfaction vs. promotion. In *In Proceedings of Beyond Personalization 2005, the Workshop on the Next Stage of Recommender Systems Research(IUI2005*, pages 13–18, 2005.

[5] 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.

[6] I. Cantador, A. Bellogín, and P. Castells. A multilayer ontology-based hybrid recommendation model. *AI Commun.*, 21:203–210, April 2008.

[7] S. Debnath, N. Ganguly, and P. Mitra. Feature weighting in content based recommendation system using social network analysis. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1041–1042, New York, NY, USA, 2008. ACM.

[8] 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.

[9] T. Franz, A. Schultz, S. Sizov, and S. Staab. Triplerank: Ranking semantic web data by tensor decomposition. In *Proc. of the 8th ISWC*, ISWC '09, pages 213–228, 2009.

[10] M. Garden and G. Dudek. Semantic feedback for hybrid recommendations in recommendz. In *IEEE Int. Conf. EEE'05*, pages 754–759, 2005.

[11] J. Golbeck and J. Hendler. Filmtrust: Movie recommendations using trust in web-based social networks. In *Proceedings of the IEEE CCNC*, 2006.

[12] H. Guo. Soap: Live recommendations through social agents. In *5th DELOS Workshop on Filtering and Collaborative Filtering.*

[13] 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.

[14] B. Heitmann and C. Hayes. Using linked data to build open, collaborative recommender systems. In *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, 2010.

[15] 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.

[16] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

[17] A. Passant. dbrec: music recommendations using dbpedia. In *Proc. of 9th Int. Sem. Web Conf.*, ISWC'10, pages 209–224, 2010.

[18] 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.

[19] 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*, Lecture Notes in Computer Science, pages 147–166. 2007.

[20] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook.* Springer, 2011.

[21] J. Salter and N. Antonopoulos. Cinemascreen recommender agent: Combining collaborative and content-based filtering. *IEEE Intelligent Systems*, 21:35–41, January 2006.

[22] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, November 1975.

[23] 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.

[24] N. Shabir and C. Clarke. Using linked data as a basis for a learning resource recommendation system. In *Learning in the Synergy of Multiple Disciplines, Proceedings of the EC-TEL 2009*, volume 5794 of *Lecture Notes in Computer Science.* Springer, October 2009.

[25] M. Srinivas and L. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17 –26, jun 1994.

[26] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. Moviexplain: a recommender system with explanations. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, pages 317–320. ACM, 2009.

[27] 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.

[28] R. Tous and J. Delgado. A vector space model for semantic similarity calculation and owl ontology alignment. In *DEXA*, pages 307–316, 2006.

[29] C. Wartena, W. Slakhorst, and M. Wibbels. Selecting keywords for content based recommendation. In *CIKM*, pages 1533–1536, 2010.

[30] R. W. White and R. A. Roth. Exploratory Search: Beyond the Query-Response Paradigm. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 1(1):1–98, Jan. 2009.