Building a relatedness graph from Linked Open Data: A case study in the IT domain

Building a relatedness graph from Linked Open Data: a case study in the IT domain

Tommaso Di Noia, Vito Claudio Ostuni, Jessica Rosati,
Paolo Tomeo, Eugenio Di Sciascio

Polytechnic University of Bari
via E. Orabona, 4 – 70125 Bari, Italy

Roberto Mirizzi

Yahoo! Inc.
701 First Avenue – Sunnyvale, CA 94089 USA

Claudio Bartolini

HP Labs
1501 Page Mill rd. – Palo Alto, CA 94304 USA

Abstract

The availability of encyclopedic Linked Open Data (LOD) paves the way to a new generation of knowledge-intensive applications able to exploit the information encoded in the semantically-enriched datasets freely available on the Web. In such applications, the notion of relatedness between entities plays an important role whenever, given a query, we are looking not only for exact answers but we are also interested in a ranked list of related ones. In this paper we present an approach to build a relatedness graph among resources in the `DBpedia` dataset that refer to the IT domain. Our final aim is to create a useful data structure at the basis of an expert system that, looking for an IT resource, returns a ranked list of related technologies, languages, tools the user might be interested in. The graph we created is a basic building block to allow an expert system to support the user in entity search tasks in the IT domain (e.g. software component search or expert finding) that goes beyond string matching typical of pure keyword-based approaches and is able to exploit the explicit and implicit semantics encoded within LOD datasets. The graph creation relies on different relatedness measures that are combined with each other to compute a ranked list of candidate resources associated to a given query. We validated our tool through experimental evaluation on real data to verify the effectiveness of

Email addresses: `{tommaso.dinoia, vitoclaudio.ostuni, jessica.rosati, paolo.tomeo, eugenio.disciascio}@poliba.it` (Tommaso Di Noia, Vito Claudio Ostuni, Jessica Rosati,
Paolo Tomeo, Eugenio Di Sciascio), `robertom@yahoo-inc.com` (Roberto Mirizzi), `claudio.bartolini@hp.com` (Claudio Bartolini)

the proposed approach.

Keywords: Relatedness graph, Linked data, DBpedia

---

1. Introduction

The emergence of the crowd computing initiative has brought on the Web a new wave of tools enabling collaboration and sharing of ideas and projects, ranging from simple blogs to social networks, as well as software platforms and even mashups. However, when these web-based tools reach the "critical mass" one of the problem that suddenly arises is how to retrieve content of interest from such rich repositories. As a way of example, we may refer to a platform to share software components, where programmers can publish APIs and mashups. When a user uploads a new piece of code, they tag it so that the component will be later retrievable by other users. Components can be retrieved through a keywords-based search or browsing across categories, most popular items or new updates. Most of the current systems usually rely on text matching between a search query and the textual description of a resource or a set of associated tags. A match is found when keywords, or patterns of keywords expressed in the query appear also in the description associated to the resource. However, text-based approaches suffer from the innate problems of ambiguity of natural language (Resnik, 1999). Even if the query and the resource descriptions are somehow structured, the same issues persist. In particular, one of the biggest deficiency in these approaches is their inability to capture the meaning of terms expressed both in the query and in the description, and the semantic relations between such terms. As an example, let us consider some cases where systems based exclusively on text analysis fail, with a particular emphasis on the IT domain:

- Both *SVM* and *Support Vector Machine* refer to the same Machine Learning algorithm. A textual approach by itself is not able to deal with synonymy.
- *Ubuntu* and *Debian* are two Linux distributions, but for a text-based system there is no way to understand how they are related.
- *PHP* and *MySQL* are two different technologies but strongly related with each other. Indeed, *MySQL* is the de facto standard DBMS used when developing *PHP* applications.
- *Java* is an *Object-oriented programming language*. The relation *isA* is pretty common when modelling knowledge domains. However, algorithms that are purely based on keywords cannot understand it.

The previous examples stress once more the importance of capturing semantic relations among entities, and of being able to identify (semantically) related resources (together with a relatedness value).

In this paper we show how to tackle such problems by considering entities and their associated semantics instead of simple keywords. In particular, we demonstrate that by leveraging knowledge bases which are freely available in the Web of Data we can compute the *relatedness* between concepts belonging to the IT (Information Technology) domain.

In fact, the notion of relatedness is wider than that of similarity. While the latter refer to specific classes of objects (the class of databases, the class of programming languages, etc.), the former refers to the whole knowledge space (database and programming languages, etc.). By remaining in the IT domain, if we consider *MySQL* and *PostgreSQL* we may say if they are similar or not as they are two DBMSs. On the other hand, if we consider *MySQL* and *PHP* we cannot state anything about their similarity but we can say if they are related with each other.

In this work we present semantic-aware measures to evaluate the relatedness values between IT concepts. Using these measures, we then build a graph where nodes are IT concepts (programming languages, databases, technologies, frameworks, etc.) and edges between nodes indicate they are related with each other. We also associate a numerical label to each edge that represents the relatedness value between two nodes. Having a tool able to measure and evaluate how much two resources are related with each other, is a key factor in the design and development of an expert system able to foster the process of selecting those resources semantically related (to different extents) to the ones that the user is looking for. Indeed, one of the main tasks an expert system must be able to cope with is that of supporting human users in decision-making processes such as "*help me in finding those items better corresponding to my needs*". Within a knowledge space, encoding the notion of relatedness among resources as a graph may, for instance, allow an expert system to: (i) support the users in ranking those resources which relate to the ones they are interested in; (ii) guide the users through an exploratory browsing (Marchionini, 2006) of the knowledge space by following links whose semantics represents the relatedness degree between the explored nodes.

Our graph is built by leveraging and combining statistical knowledge obtained from the Web and semantic knowledge extracted from the encyclopedic knowledge base `DBpedia`[1]. We adopt an approach based on machine learning to effectively combine such information. The approach we present here builds on top of (Mirizzi et al., 2010). Nevertheless, there are many differences and improvements. First of all, they rely on a very expensive, from a computational point of view, process for extracting relevant resources from `DBpedia`. It considers a continuous interaction between the graph exploration and the computation of Web-based conditional probabilities. Moreover, they manually combine different features in a naive way instead of automatically combining them as we propose here.
The novel contributions of this work are listed in the following:

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

- proposal of a measure for finding the relatedness of concepts in the IT domain, based on statistical, textual and semantic analysis combined via both a Learning to Rank (Liu, 2009) (LTR) and a data fusion (Nuray and Can, 2006) approach;
- construction of a relatedness graph for IT concepts on top of `DBpedia`;
- experimental evaluation of the approach on real data extracted from job posts.

The remainder of this paper is structured as follows. In the next section we give a brief overview of the Semantic Web technologies we adopt in our approach. In Section 2 we describe the advantages of using semantic knowledge bases and we provide information about `DBpedia`. The relatedness measure between IT terms and the graph building are detailed in Section 3. In Section 4 we present the results of the evaluation of our approach. Related work is discussed in Section 5. Conclusion and Future work conclude the paper.

2. Linked Data as a knowledge source in the IT domain

If we wanted to build an expert system able to catch relatedness between IT technologies and tools we would have needed a way to capture the meaning behind keywords in order to overcome the issues of text-based approaches. Indeed, in this knowledge-intensive scenario, detailed information about entities plays a fundamaental role. During the last few years, the Web has been evolving in the so called Web of Data where the main actors are no more pages identified by a URL but resources/data identified by a URI. In this transformation process the Linked Open Data (LOD) (Bizer et al., 2009a) initiative has been a first calls citizen. The *Linking Open Data* community project started in 2007 with the goal of augmenting the current Web with data published according to Semantic Web standards. The idea is to use `RDF`[2] to publish various open datasets on the Web as a vast decentralized knowledge graph, commonly known as the LOD cloud. As of today, several dozen billion `RDF` triples are freely available covering diverse knowledge domains and tightly connecting different datasets with each other.

DBpedia. One of the most popular datasets in the LOD compass is `DBpedia` (Bizer et al., 2009b). It is a community effort to extract structured information from Wikipedia and make it freely accessible as `RDF` triples. This knowledge base currently describes 4 million resources, out of which 3.22 million are classified in a consistent ontology[3]. Its `SPARQL` endpoint[4] allows anyone to ask complex queries about such resources. Each element in `DBpedia` is identified by its own URI in order to avoid ambiguity issues. For example, the programming language *Java* is referred to as the resource identified by the URI

[2] See Appendix A for a quick overview on `RDF` and `SPARQL`.
[3] `http://wiki.dbpedia.org/Ontology`
[4] `http://dbpedia.org/sparql`

dbpedia:Java_(programming_language), whereas the software platform *Java* is identified by the URI dbpedia:Java_(software_platform). The resource dbpedia:Java_(disambuigation) describes all the possible meanings for the label *Java* thanks to the property dbpedia-owl:wikiPageDisambiguates. Similarly, both the URI dbpedia:Svm_(machine_learning) and the analogous URI dbpedia:Support_vector_machine refer to the same Machine Learning algorithm, hence to the same resource. In DBpedia this relation is captured via the property dbpedia-owl:wikiPageRedirects that connects the former to the latter entity.

Compared to other hierarchies and taxonomies, DBpedia has the benefit that each term/resource is endowed with a rich textual description via the dbpedia-owl:abstract property and at least one textual label via rdfs:label. The value associated to dbpedia-owl:abstract is a string containing the text before the table of contents (at most 500 words) of a Wikipedia page, while the property rdfs:label contains the title of the Wikipedia page. The multilingual nature of Wikipedia is reflected in the values of dbpedia-owl:abstract and rdfs:label. In fact, given a DBpedia URI, we may have a description and a label for each language available in the corresponding Wikipedia page. This is particularly interesting when one wants to build a tool fed by the information represented in DBpedia since it allows the development of natively multi-lingual applications. DBpedia also maps hypertextual links between Wikipedia pages by the property dbpedia-owl:wikiPageWikiLink. For any link connecting a Wikipedia document d_i to another Wikipedia document d_j, DBpedia asserts a corresponding dbpedia-owl:wikiPageWikiLink from u_i to u_j, where u_i and u_j are the DBpedia URIs corresponding to the Wikipedia pages d_i and d_j, respectively. Wikipedia authors organize articles by topics, clustering them into categories, with the purpose of grouping together articles on related topics. Categories may contain several subcategories, i.e., categories that are more specific than their parents. In DBpedia, the relations between categories and articles are maintained by using the Dublin Core Metadata Element Set[5] (often referred to as DCTERMS) and the Simple Knowledge Organization System[6] (SKOS) vocabularies. In particular, the dcterms:subject property relates a resource to its corresponding Wikipedia category, while the skos:broader property links a category to its super-categories. For example, the resource dbpedia:Java_(programming_language) is connected to the category dbpedia:Category:Object-oriented_programming_languages by means of the property dcterms:subject. Similarly, the specific category dbpedia:Category:Object-oriented_programming_languages is connected to its more general category dbpedia:Category:Programming_languages by means of the property skos:broader. It is worth noticing that category-based information, although not very rigorous from an ontological point of view, is a very useful tool to group together resources in an automated way. Moreover, the categorization of resources is very rich and fine grained. Finally, the rdf:type

[5] http://purl.org/dc/terms/subject
[6] http://www.w3.org/2004/02/skos#

property is used to assign a resource to a class, as defined in the DBpedia Ontology. For example, the resource dbpedia:Java_(programming_language) is an instance of the class dbpedia-owl:ProgrammingLanguage. This latter is rdfs:subClassOf the class dbpedia-owl:Software. We observe that the lack of depth in the hierarchical structure of the DBpedia ontology in its current version does not allow for precise and accurate classification of RDF entities.

DBpedia changes as Wikipedia changes, resulting in a continual updates of its triples not requiring human intervention.

In summary, the main advantages of leveraging LOD datasets as DBpedia in knowledge-based applications are:

- a huge amount of structured information on heterogeneous knowledge domains freely available and ready to be processed;
- no ambiguity issues typical of keywords thanks to item representations by unique identifiers (URIs);
- classification of resources via both categories and ontology classes;
- Semantic Web standards to query the knowledge base;
- rich multi-domain and multi-language data;
- avoidance of version control issues due to the continual synchronization with Wikipedia.

3. Computing a semantic relatedness graph in the IT domain as a ranking problem

As argumented in Section 1, in the IT domain, very often it is fundamental to understand both *if* and *how much* two entities relate with each other. Relatedness is a more general concept than similarity (Resnik, 1995). Let us clarify this with an example. If we consider the two DBpedia entities *Django*[7] and *Python*[8], the relation between them is that the former (a web framework), is developed by using the latter (a programming language). Obviously, the two entities are *related* with each other even if they are not *similar*. As another example, let us consider the two resources *Java*[9] and *C++*[10]. They are related with each other because they are both object-oriented programming languages. In this case they are both somehow similar and related.

Computing relatedness between pairs of resources can be a very time-consuming task as it often requires not just pure content-based information (as text descriptions) but also data coming from external sources (as number of incoming and

[7] http://dbpedia.org/resource/Django_(web_framework)
[8] http://dbpedia.org/resource/Python_(programming_language)
[9] http://dbpedia.org/resource/Java_(programming_language)
[10] http://dbpedia.org/resource/C++

outcoming links for a Web page). This is the main reason why the computation is usually done offline and relatedness results are stored in specific data structures such as (knowledge) graphs. They are labelled graphs where the label of an edge indicates how much two connected nodes are related with each other.

The semantic relatedness graph we build, with reference to the IT domain, exploits the knowledge encoded in `DBpedia` both to identify entities in a unique way and to compute the related nodes based on a neighbourhood semantic exploration. In our setting we formulate the relatedness measure as a ranking problem. Given a resource that we consider as query, we rank the most related resources with respect to it. Specifically (see Section 3.2), we combine different ranking features by adopting either a Learning to Rank (Liu, 2009) or an unsupervised data fusion approach (Nuray and Can, 2006). The ranking features come from different knowledge sources:

- `DBpedia` graph-based ranking;
- Wikipedia text-based ranking;
- Web-based ranking.

The rationale behind this combination is to obtain a robust measure leveraging strengths and mitigating weaknesses of each feature. In particular:

- the `DBpedia` graph-based ranking allows us to exploit explicit semantic connections between resources/skills, but they could not be enough to assess the relatedness between resources;
- the textual approach adopted in the Wikpedia text-based ranking allows the system to discover connections between resources based on common keywords, but the meaning behind such keywords is not exploited;
- the Web-based ranking provides a statistical hint about the relatedness between resources which bases on their popularity and on their co-occurrence on the Web, but it lacks of an explicit semantics.

Our relatedness graph is a directed labelled graph represented by the 3-tuple $G = \langle S, E, L \rangle$ where S is the set of vertices, E represents the set of directed edges between pair of nodes in S and $L = \{1, \ldots, N\}$ is the set of labels we assign to edges in E. Given two nodes s_q (for query) and s_k connected by a directed edge $e_{q,k}$, the label $l_{q,k}$ indicates the ranking position of s_k in the list of resources related to s_q. As the notion of relatedness is symmetric, for each pair s_q and s_k we also have an edge $e_{k,q}$ whose label $l_{k,q}$ states the position of s_q in the list of resources related to s_k. In Figure 1 we show an example of the relatedness graph we build. There, we see that s_2 is ranked as 1 in the list of resources related to s_1 while it is ranked as 6 in the list of s_3. Moreover, we have s_1 ranked as 2 for s_2.

In order to build G we first retrieve from `DBpedia` all the entities related to the IT domain thus representing the set of vertices S. Then, for each $s_q \in S$ we identify a list $\vec{s} = \langle s_k \mid \text{ with } k = 1, \ldots, M \rangle$ of related entities. The set E is

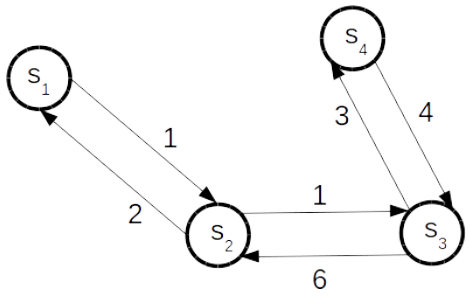

Figure 1: An example of a relatness graph in our framework.

composed by all the edges $e_{q,k}$ going from s_q to each element $s_k \in \vec{s}$. Finally, we assign a label $l_{q,k} \in L$ to $e_{q,k}$ that correspond to the position (ranking) of s_k in the list $\vec{s}$.

ITJobsWatch. The model we present, needs some parameters to be set either in the computation of features values and while selecting relevant resources from `DBpedia`. To train the model we adopted a supervised approach by exploiting `ITJobsWatch`[11] as an external data source for obtaining ground truth relatedness values. `ITJobsWatch` is a website that provides a detailed board of the UK information technology job market. Data are continuously updated from multiple IT recruitment websites. The purpose of the project is to present a compact and accurate snapshot of the job market conditions in the UK. In particular, the website allows users to: (1) know what IT skills are requested on the job market for certain job titles, and in what measure, (2) determine the average salary associated to IT skills, (3) discover how IT skills are interrelated with each other. For the purpose of our work, the latter aspect is the most interesting one. In fact, on `ITJobsWatch` each skill is associated with a ranked list of *related* skills. Here, the notion of *relatedness* between the two skills s_q and s_k derives from their co-occurrence within job posts: the percentage of IT job posts within the UK citing skill s_q that also mentioned skill s_k, during a time frame of six months. For example, if we look at the skills related to *Java*[12], we find *Spring* and *J2EE* ranked in the first ten positions, with a **relative co-occurrence value** of 23.31% and 19.54%, respectively. In other words, on the average, every 100 job posts citing *Java*, about 23 also cite *Spring* and 20 cite *J2EE*. Being these ranked lists obtained from real job posts, they allow us to get an accurate picture of job demand.

[11] `http://www.itjobswatch.co.uk`

[12] `http://www.itjobswatch.co.uk/jobs/uk/java.do`, page accessed on February 11, 2014.

3.1. Collecting IT resources via DBpedia exploration

As we said before, in order to build the graph G, the first step is to identify the set of vertices S, i.e. the entities belonging to our knowledge domain. In our approach, we collect them by extracting from `DBpedia` all the resources which are relevant to the IT domain through the exploration of the underlying `RDF` graph via `SPARQL` queries. In particular, the process starts by selecting a set SN of initial resources (we call them *seed nodes*) belonging to the IT domain (e.g., some programming languages, some databases, etc.) from the set DB representing all the `DBpedia` resources. Then, `SPARQL` is used to extract from `DBpedia` the resources related to each seed. Given a set of seeds $sn \in SN$ we select the set C of the most relevant categories[13] connected to sn via `dcterms:subject` and then we extract all the resources that belongs to each category $c \in C$. In order to compute how relevant a category is with respect to SN we use the information content $I(c)$ (Ross, 1976) associated to each $c \in C$ and we consider only those categories whose $I(c)$ value is greater or equal than the average. More formally, the set of relevant categories for the domain is defined as

$$\tilde{C} = \left\{ c \in C \mid I(c) \geq \frac{\sum_{c \in C} I(c)}{|C|} \right\}$$

where the information content $I(c)$ associated to the category c is computed as

$$I(c) = -log_2 p_c$$

being p_c the probability that c classifies elements belonging to SN.

$$p_c = \frac{|\{sn \texttt{ dcterms:subject c} \mid sn \in SN\}|}{|\{s \texttt{ dcterms:subject c} \mid s \in DB\}|}$$

The set S of vertices can then be formally defined as

$$S = \{s \mid s \texttt{ dcterms:subject } c \wedge c \in \tilde{C}\}$$

If we consider, for instance the set $SN = \{$`dbpedia:PHP`, `dbpedia:MySQL`, `dbpedia:Oracle_Database`, `dbpedia:F_Sharp_(programming_language)`, `dbpedia:JavaScript`, `dbpedia:C++`$\}$ we obtain the set $\tilde{C}$ composed by the categories represented in Table 1[14]. We observe that by considering $\tilde{C}$ instead of the whole C, we filter out non relevant categories such as:
`dbpedia:Category:American_inventions`
`dbpedia:Category:Cross-platform_free_software`
`dbpedia:Category:Filename_extensions`

[13] While extracting relevant elements, we do not consider ontological information encoded via `rdf:type` statements because, as discussed in Section 2, the `DBpedia` ontology is very shallow thus allowing a very coarse grained classification of resources.

[14] From the set C we filtered out all those categories containing a year such as `dbpedia:Category:Programming_languages_created_in_1983`. They can be easily identified as their name follows a precise pattern.

dbpedia:Category:Client-server_database_management_systems
dbpedia:Category:Web_programming
dbpedia:Category:Linux_database-related_software
dbpedia:Category:High-level_programming_languages
dbpedia:Category:Object-based_programming_languages
dbpedia:Category:Algol_programming_language_family
dbpedia:Category:ML_programming_language_family
dbpedia:Category:Class-based_programming_languages
dbpedia:Category:MySQL
dbpedia:Category:Statically_typed_programming_languages
dbpedia:Category:Prototype-based_programming_languages
dbpedia:Category:.NET_programming_languages
dbpedia:Category:JavaScript
dbpedia:Category:Relational_database_management_systems

Table 1: An example of $\tilde{C}$ computed from a small set of seed nodes. Categories are listed in decreasing order of $I(c)$.

In our setting, we selected SN as the set of all the resources extracted from `ITJobsWatch` and mapped to `DBpedia`. In particular, with respect to `ITJobsWatch` classification, we concentrated on *Processes & Methodologies*, *Applications*, *Libraries, Frameworks & Software Standards*, *Programming Languages*, *Database & Business Intelligence*, *Development Applications* and *Systems*. At the end of the mapping process we had 941 IT resources mapped to `DBpedia`.

Once we compute S by performing all the queries for each seed node, we obtain the vocabulary of IT resources needed to build the relatedness graph and we are then ready to compute the relatedness between pairs of IT resources. A basic approach to create the knowledge graph could be: for each pair of nodes belonging to the IT vocabulary S compute a relatedness value. The simple approach is not very efficient since we also check nodes with a very low (almost zero) relatedness value, such as *HTML* and *Object-Orientd Programming*. Our hypothesis is that related nodes are connected within the `DBpedia` graph via a path up to a maximum length D and nodes connected by a path whose length is greater than D are not semantically related. Moreover not all the paths in `DBpedia` are relevant in order to discover relatedness relations within the IT domain. There are properties that are used more frequently in the domain of interest, such as `dbpedia-owl:programmingLanguage` or `dbpedia-owl:computingPlatform`. Therefore, we initially looked for the set of most frequent properties used with the nodes in S and then, based on these properties we explored `DBpedia` to evaluate D. Given a node $s_q \in S$, we evaluate the occurrence of properties used as *predicate* in the `RDF` triples having s_q either as *subject* or as *object*. We collected a total of 266 different properties. As shown in Figure 2, there are many properties with a very low occurrence value (195 properties out of 266 occurred less than 10 times). For the exploration of the `DBpedia` graph we then

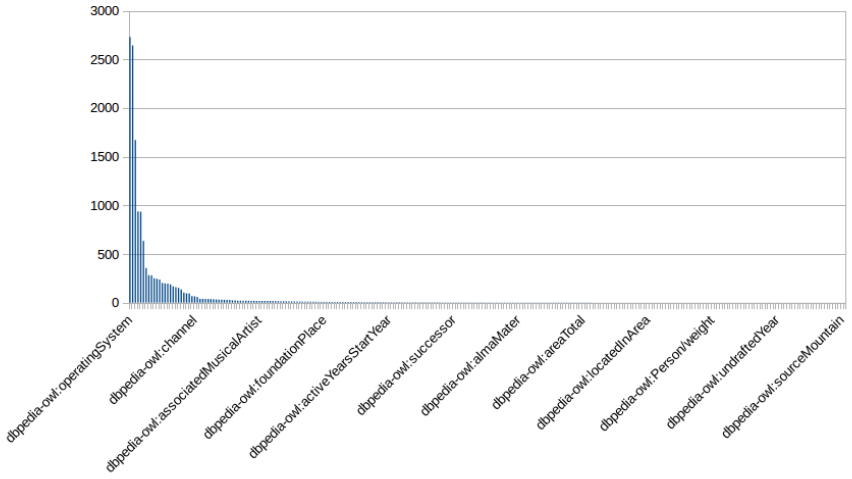

Figure 2: Distribution of `DBpedia` properties in the IT domain

`http://dbpedia.org/ontology/operatingSystem`
`http://dbpedia.org/ontology/programmingLanguage`
`http://dbpedia.org/ontology/computingPlatform`
`http://dbpedia.org/ontology/industry`
`http://dbpedia.org/ontology/genre`
`http://dbpedia.org/ontology/wikiPageWikiLink`
`http://purl.org/dc/terms/subject`
`http://www.w3.org/2004/02/skos/core#broader`

Table 2: Properties used to explore the DBpedia graph

considere as relevant: (i) the top 5 (see Figure 3) extracted properties; (ii) the `dbpedia-owl:wikiPageWikiLink`; (iii) the properties used to encode category-based information, i.e., `dcterms:subject` and `skos:broader`. The whole set of adopted properties is represented in Table 2. From now on, we will refer to this set as P. The computation of the maximum exploration depth D relies on some information coming from `ITJobsWatch`. In particular, for each resource $s_q^{\texttt{ITJobsWatch}} \in SN$ we retrieved the *Top 30 Related IT Skills*[15]. Then we consider all the resources $s_k^{\texttt{ITJobsWatch}}$ whose *relative co-occurrence value* is greater than the average value and we check in `DBpedia` the length of the shortest paths between the corresponding resources $s_q^{\texttt{ITJobsWatch}}$ and $s_k^{\texttt{ITJobsWatch}}$ by considering only those paths composed by properties in P. The distribution of the shortest

[15]See `http://www.itjobswatch.co.uk/jobs/uk/django.do` for an example.

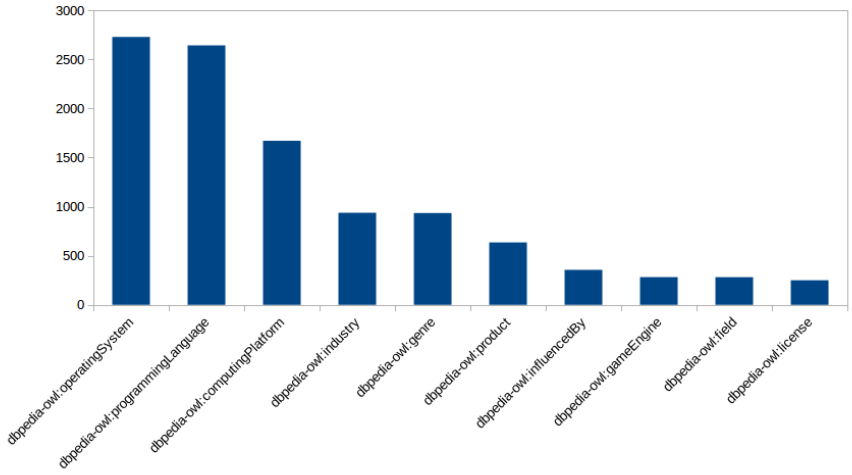

Figure 3: Top-10 most frequent properties in the IT domain

hops	number of shortest paths
1	840
2	2677
3	1727
4	124

Table 3: Distribution of the shortest paths in `DBpedia` between entities retrieved from `ITJobsWatch` considering properties in P

paths is shown in Table 3. We observe that in the worst case we have a path of length **4**. We thus set $D = 4$ as we consider it as the maximum number of hops in the `DBpedia` graph needed to connect two resources belonging to the IT domain. Hence, for each resource $s_q \in S$ we perform a depth-first exploration of the graph and we save in a list $\vec{s}$ all the nodes reachable in at most 4 hops. We call these nodes *neighbours of* s_q. When the exploration stops we can eventually compute the relatedness values between s_q and all the elements in $\vec{s}$ by means of the different ranking features as described in the following. It is noteworthy that the exploration is performed by following only the set of properties P shown in Table 2. As we are interested in relatedness between pairs of resources we consider the `DBpedia` graph as undirected. For example, if we consider the resource `dbpedia:PHP` $\in S$ and the property `dbpedia-owl:wikiPageWikiLink` $\in P$, the corresponding query is

```
SELECT DISTINCT ?r
WHERE {
 {dbpedia:PHP dbpedia-owl:wikiPageWikiLink ?r .}
```

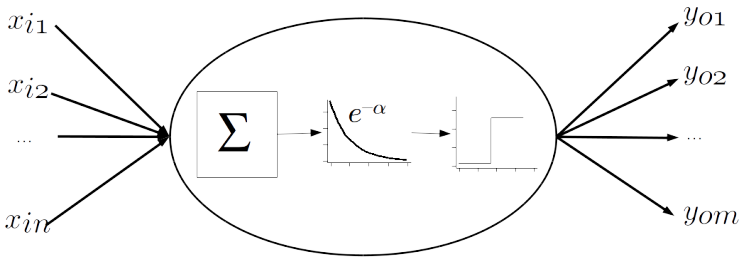

Figure 4: Spreading activation node model.

```
 UNION
 {?r dbpedia-owl:wikiPageWikiLink dbpedia:PHP}
}
```

At the end of the exploration, for our dataset, we have lists $\vec{s}$ whose length goes from 4 to 127. This means that for each resource s_q we will have at least 4 and at most 127 related nodes in the final knowledge graph.

3.2. ***Ranking features***

As introduced at the beginning of this section, we compute the final rank by combining diverse ranking features. A graph-based, a text-based and a Web-based ranking feature are combined to return the aggregate ranking list $\vec{s}_{aggr}$ computed with respect to a resource s_q used as query.

3.2.1. ***Graph-Based ranking***

The graph-based ranking is based on the *spreading activation* algorithm (Crestani, 1997).

Spreading activation has a history of applications in cognitive psychology, artificial intelligence and, more recently, in information retrieval. The spreading activation theory of human semantic processing was first proposed by Quillian (Quillian, 1968). This model was introduced for semantic networks of interlinked concepts which contain the basic elements we find today in `RDF` graphs (as the one represented by `DBpedia`). The processing technique behind spreading activation is rather simple, consisting of one or more propagated impulses and a termination check. In addition, the model can implement propagation decays and constraints on the spreading activation process. The process starts with an initial energy impulse that propagates from the input query node – an IT resource in our case – to its neighborhood nodes. Iteratively, these nodes propagate part of their energy to their neighborhood nodes till a convergence criterion is reached. Alternatively to or in combination with the convergence criterion, a maximum number of iterations can be set as stop criterion.

More in detail, a generic node s_k receives energy impulses from its in-links, then it accumulates, processes and propagates this energy to its out-links following the model illustrated in Fig. 4. The values $x_{i_1}, .., x_{i_n}$ represent the energy

impulses coming from the in-links that could potentially activate the current node. The complete model also takes into account a decay factor (usually an exponential function) and a step activation function. So, the actual activation energy is $x = e^{-\alpha} \cdot \sum_j x_{i_j}$. If x is greater or equal to a threshold value T associated to the step function then the node propagates x by equally splitting the energy among the out-links. The aim of the decay factor is to favour closer connections to farther ones over the network in case the exploration goes deep in the network. The resulting output energy values $y_{o_1}, .., y_{o_m}$ become input energy for the next nodes while the output value x represents the relatedness between the input query node s_q and the current node s_k. The process terminates either when there are no active nodes which propagate their energy to the followers or when a maximum number MAX of iterations is reached.

In our implementation, based on the results related to the longest short path presented in the previous section, we adopt a spreading activation with a maximum number of iteration $MAX = D$. The maximum distance covered by energy coming from s_q to reach s_k is then 4. As this is a quite small value we assume the decay factor as negligible and set $\alpha = 0$. Since we adopt a spreading activation with a maximum number of iterations, we chose to set the activation threshold $T = 0$. Hence, the final formulation of the relatedness score obtained by the spreading activation referred as $sa(s_q, s_k)$ is:

$$sa(s_q, s_k) = \sum_j x_{i_j}$$

where x_{i_j} is the portion of energy starting from s_q and arriving in s_k.

3.2.2. ***Text-based ranking***

In text-based ranking we exploit *text* and *link analysis* in `DBpedia`. in particular, we look at the objects of the triples involving the properties `dbpedia-owl:abstract` and `dbpedia-owl:wikiPageWikiLink`, respectively. We remember that the former is a datatype property whose object is a string containing the first paragraph of the corresponding Wikipedia page. The latter is an object property whose object is a `DBpedia` resource that gets an incoming hyperlink on the corresponding Wikipedia page.

Given two `DBpedia` nodes s_q and s_k, we check if the label of node s_q is contained within the abstract of node s_k, and vice versa. The main assumption behind this check is that if a `DBpedia` resource name appears in the abstract of another `DBpedia` resource, then it is reasonable to think that the two resources are related with each other. Specifically, we check if the `rdfs:label` of s_q is contained within the `dbpedia-owl:abstract` of s_k (and vice versa). In case the value of `rdfs:label` is composed by more than one single word, we also consider all possible n-grams.

Let Q and K be the number of words composing the label of s_q and s_k, respectively. Let us denote by N_q the number of n-grams composing the label of s_q which are also in the abstract of s_k and by N_k the number of n-grams composing the label of s_k which are also in the abstract of s_q. We define the

text-based similarity between s_q and s_k as:

$$abstract(s_q, s_k) = \frac{1}{2} \cdot \left(\frac{N_q}{\sum_{i=1}^{Q} i} + \frac{N_k}{\sum_{i=1}^{K} i} \right)$$

We see that $abstract(s_q, s_k) \in [0, 1]$.

Similarly to what we do with labels and abstracts, we also check if the Wikipedia page of resource s_q has an outgoing link to the Wikipedia page of resource s_k, and vice versa. If `DBpedia` contains the `RDF` triple s_q `dbpedia-owl:wikiPageWikiLink` s_k (and/or s_k `dbpedia-owl:wikiPageWikiLink` s_q), we assume a stronger relation between the two resources. We evaluate the strength of the connection as follow:

$$wiki(s_q, s_k) \quad = \quad \begin{cases} 0, & \text{no } \texttt{dbpedia-owl:wikiPageWikiLink} \text{ between } s_q \text{ and } s_k \text{ in } \texttt{DBpedia} \\ 1, & s_q \texttt{ dbpedia-owl:wikiPageWikiLink } s_k \text{ exists in } \texttt{DBpedia} \\ 1, & s_k \texttt{ dbpedia-owl:wikiPageWikiLink } s_q \text{ exists in } \texttt{DBpedia} \\ 2, & \text{both } s_q \texttt{ dbpedia-owl:wikiPageWikiLink } s_k \text{ and} \\ & s_k \texttt{ dbpedia-owl:wikiPageWikiLink } s_q \text{ exist in } \texttt{DBpedia} \end{cases}$$

3.2.3. ***Web-based ranking***

Given two `DBpedia` resources s_q and s_k, we verify how many Web pages contain (or have been tagged by) the value of the `rdfs:label` associated to s_q and s_k. Then, we compare these values with the number of pages containing (or tagged by) both labels. The Web pages count is obtained by querying the following information sources: *Google*, *Bing* and *Delicious*. We select more than one search engine because we do not want to bind the result to a specific algorithm. Moreover, we want to rank a resource not only with respect to the popularity of related pages on the Web, but also considering the popularity of such resources among users (using *Delicious*, a popular social tagging system). In this way we are able to combine two different perspectives on the popularity of a resource: the one related to the words occurring within Web documents, the other one exploiting the social nature of the current Web.

We compute the relatedness of two resources s_q and s_k with respect to a given external information source $\sigma \in \{G, B, D\}$[16] as:

$$web^{\sigma}(s_q, s_k) = \frac{1}{2} \cdot \left(\frac{d_{s_q, s_k}}{d_{s_q}} + \frac{d_{s_q, s_k}}{d_{s_k}} \right)^{\sigma}$$

where d_{s_q} and d_{s_k} represent the number of documents containing (or tagged by) the `rdfs:label` associated to s_q and s_k respectively, and d_{s_q, s_k} represents how many documents contain (or have been tagged by) both the label of s_q and s_k. The formula is symmetric and the returned value is in $[0, 1]$. It is noteworthy that the first term in $web^{\sigma}(s_q, s_k)$ represents the conditional probability

[16] G is used to represent the information source Google, B for Bing and D for Delicious.

$p^{\sigma}(s_k|s_q)$ while the second is $p^{\sigma}(s_q|s_k)$.
Let us clarify this relatedness measure by means of an example and consider s_q = `dbpedia:Python_(programming_language)` and s_k = `dbpedia:Django_(web_framework)`. The `rdfs:label` for s_q is *Python (programming language)* and for s_k is *Django (web framework)*. On Google[17], $d_{s_k} = 2e7$ and $d_{s_q} = 3e6$. For the conjunct query *Python (programming language) Django (web framework)*, $d_{s_q,s_k} = 1e6$. Given these values, it is straightforward to compute the score web^G(`dbpedia:Python_(programming_language)`, `dbpedia:Django_(web_framework)`).

3.3. ***Ranking features aggregation***

Once we compute the different ranking features, we aggregate them to obtain an effective and accurate ranking of IT resources. Here we present two approaches for feature aggregation: the one based on Learning to Rank (Liu, 2009), the other exploiting techniques coming from the Data Fusion field (Nuray and Can, 2006). The former employs a supervised approach to learn a ranking function, in particular we use the Ranking SVM algorithm (Joachims, 2002). The latter is a unsupervised method based on a popular voting technique called Borda count (Dwork et al., 2001).

query	**res**	*sa*	*abstract*	*wiki*	web^G	web^B	web^D	*y*
Java	*Spring*	0.65	0.5	2	0.21	0.13	0.23	0.33
Java	*SQL*	0.58	0.2	0	0.45	0.75	0.35	0.24
Java	*Oracle*	0.43	0.33	1	0.32	0.80	0.65	0.23
Java	*J2EE*	0.45	0.75	1	0.22	0.13	0.19	0.20

Table 4: An example of the structure of the dataset for the query *Java*.

3.3.1. ***Ranking SVM algorithm***

The goal of using Learning to Rank in our approach is to automatically learn a ranking function from training data to obtain a model able to sort IT resources according to their relatedness with respect to a query. Training data is represented as a set of input-output pairs $(\vec{x}_{s_q,s_k}, y_{s_q,s_k})$, where $\vec{x}_{s_q,s_k} \in \mathbb{R}^N$ (N is the number of features) is a feature vector (Fuhr, 1989) that for each single feature encodes the match between the query s_q and the IT concept s_k, while y_{s_q,s_k} is a rank label, such as the rank position or relevant/irrelevant in case of binary data. Table 4 shows an example of the structure of the dataset just described. In this case s_q – the query concept – is the resource *Java* and the IT concepts to rank are: *Spring*, *SQL*, *Oracle*, *J2EE*. Each component of $\vec{x}_{Java,s_k}$ encodes the relatedness between each resource and *Java*. For example for $s_k = Spring$ we have:

$$\vec{x}_{Java,Spring} = \langle 0.65, 0.5, 2, 0.21, 0.13, 0.23 \rangle$$

[17]Results accessed on Feb 9th, 2014.

where the value 0.65 represents the relatedness for the pair (*Java*, *Spring*) obtained with the spreading activation method, and the other values refer to the other features, as detailed in Section 3.2. The last column represents the relatedness score y_{s_q,s_k} given by `ITJobsWatch` in the *Top 30 Related IT Skills* and we consider it as ground truth data. We recall that we are not interested in the relatedness score in itself but in the ranking induced by it.

In our work we adopt Ranking SVM (Joachims, 2002), that is a pairwise approach to the Learning to Rank problem. The basic idea of Ranking SVM is to formalize Learning to Rank as a problem of binary classification on instance pairs, and then to solve the problem using Support Vector Machines.
The goal is to optimize a scoring function $f(\vec{x}_{s_q,s_k}) = \vec{w} \cdot \vec{x}_{s_q,s_k}$ that maps $\vec{x}_{s_q,s_k}$ to a labelling value y_{s_q,s_k} indicating the ranking score. If $f(\vec{x}_{s_q,s}) > f(\vec{x}_{s_q,s_j})$, then s should be ranked higher than s_j for the query s_q (or equivalently $y_{s_q,s} > y_{s_q,s_j}$).

Specifically, Ranking SVM minimizes the number of discordant pairs resolving the following optimization problem:

$$\min_{\vec{w},\xi_{q,i,j}} \quad \frac{1}{2}\vec{w} \cdot \vec{w} + \Gamma \sum \xi_{q,i,j}$$

$$\vec{w} \cdot (\vec{x}_{s_q,s} - \vec{x}_{s_q,s_j}) \geq 1 - \xi_{q,i,j} \qquad \text{if } y_{s_q,s_i} > y_{s_q,s_j}$$

$$\forall q \forall i \forall j : \xi_{q,i,j} \geq 0$$

where Γ is a parameter that allows us to trade-off margin size against training error and $\xi_{q,i,j}$ are slack variables introduced to approximate the solution. It can be shown that the problem is convex, has no local optima and is equivalent to an SVM classifier on pairwise difference vectors $\vec{x}_{s_q,s} - \vec{x}_{s_q,s_j}$ (Joachims, 2002). In our scenario, the model has been trained by considering, for each resource $s_q \in SN$, the *Top 30 Related IT Skills* with the corresponding relative co-occurrence value as $y_{s_q,s}$.

Since the linear function $f(\vec{x}_{s_q,s})$ learned is a linear combination of the feature vectors, this makes it possible to use kernel functions and extend the SVM algorithm to non-linear retrieval functions. For ranking SVM we used the implementation SVMRank[18] (Joachims, 2006). We tested different kernel functions and we obtained the best results with the RBF kernel (Schlkopf et al., 1997).

3.3.2. ***Borda count algorithm***

Borda count is a positional method, as it assigns a score corresponding to the positions in which a candidate result appears within each voter's ranked list of preferences, and the candidates are sorted by their total score. An advantage of positional methods is that they are computationally very easy: they can be implemented in linear time. Moreover, no training phase is needed.

[18] `http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`

The Borda count works as follows. Each voter ranks a fixed set of c candidates in order of preference. For each voter, the top ranked candidate is given c points, the second ranked candidate is given $c-1$ points, and so on. If there are some candidates left unranked by the voter, the remaining points are divided evenly among the unranked candidates. The candidates are ranked based on total points, and the candidate with the greatest number of points wins the election.

There is a direct analogy between multi-candidate election strategies and the problem of rank aggregations: in our case, the set of related resources are the candidates, and the different ranking features are voters expressing preferences among these candidates. No relevance scores are required, no training data is required, and the combination algorithm is simple and efficient. In our scenario considering that a single ranking can have equally preferred candidates, we adapt the classic Borda count setting. In fact, we give equal points to equally preferred candidates. If there are i equally preferred candidates starting from position $c-k$ to $c-k-i+1$ we give $\frac{1}{i}\sum_{j=0}^{i-1}(c-k-j)$ points to these i candidates.

Given the query concept s_q, the resources $\vec{s} = \langle s', s'', \ldots s^c \rangle$ to be ranked with respect to s_q and the set of features $\mathcal{X}$, we compute the points $\vec{\beta}_x(s_q, \vec{s})$ for each feature $x \in \mathcal{X}$ and then we sum the corresponding values in the vectors.

$$
\begin{aligned}
borda(s_q, \vec{s}, \mathcal{X}) &= \sum_{x \in \mathcal{X}} \vec{\beta}_x(s_q, \vec{s}) \\
\vec{\beta}_x(s_q, \vec{s}) &= \begin{pmatrix} b'_x \\ b''_x \\ \ldots \\ b^c_x \end{pmatrix}
\end{aligned}
$$

Where $b'_x, b''_x, \ldots, b^c_x$ represent the points assigned to $s', s'', \ldots, s^c$ because of the feature x. As an example, if we consider the data in Table 4 we have $s_q = Java$, $\vec{s} = \langle Spring, SQL, Oracle, J2EE \rangle$, $\mathcal{X} = \{sa, abstract, wiki, web^G, web^B, web^D\}$ and the corresponding points:

$$
\begin{aligned}
\vec{\beta}_{sa}(Java, \vec{s}) &= \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}, \quad \vec{\beta}_{abstract}(Java, \vec{s}) = \begin{pmatrix} 3 \\ 4 \\ 1 \\ 2 \end{pmatrix} \\
\vec{\beta}_{wiki}(Java, \vec{s}) &= \begin{pmatrix} 4 \\ 2.5 \\ 1 \\ 2.5 \end{pmatrix} \quad \vec{\beta}_{web^G}(Java, \vec{s}) = \begin{pmatrix} 1 \\ 2 \\ 4 \\ 3 \end{pmatrix} \\
\vec{\beta}_{web^B}(Java, \vec{s}) &= \begin{pmatrix} 1.5 \\ 1.5 \\ 3 \\ 4 \end{pmatrix}, \quad \vec{\beta}_{web^D}(Java, \vec{s}) = \begin{pmatrix} 2 \\ 1 \\ 3 \\ 4 \end{pmatrix}
\end{aligned}
$$

The final Borda count is then evaluated as:

$$borda(Java, \vec{s}, \mathcal{F}) = \begin{pmatrix} 15.5 \\ 15 \\ 14 \\ 16.5 \end{pmatrix}$$

4. Evaluation

In order to evaluate the effectiveness of our approach, we compared the results obtained by our aggregated ranking methods both with the most relevant relatedness metrics we adopt and other two metrics available in the LOD literature.

Dataset description. We evaluated our approach – and also trained it in case of the supervised algorithm Ranking SVM – on a dataset extracted from `ITJobsWatch`. The dataset is composed by 4,765 relatedness values between pairs of resources. In particular, there are 535 ranking list with 5 to 10 items.

Tested algorithms. We tested and compared the following relatedness measures:

- **Ranking SVM**. It is the supervised algorithm Ranking SVM described in Section 3.3.1.
- **Borda count**. It is the unsupervised algorithm described in Section 3.3.2.
- **Sum**. All the scores coming from the different ranking features as described in Section 3.2 are summed.
- **Random**. The ranking lists are generated randomly. We consider the Random predictor as baseline.
- **Spreading Activation**. This is the score obtained considering only the graph-based rank (sa) (see Section 3.2.1).
- **DBpediaRanker**. This is the approach presented in (Mirizzi et al., 2010). Here, only text-based (*abstract* and *wiki*) (see Section 3.2.2) and web-based (web^{σ}) (see Section 3.2.3) ranking features are considered and the final relatedness is the sum of the two ranking features.
- **Web-based**. We consider only the ranks obtained with the Web-based ranking feature (web^{σ}) (see Section 3.2.3).
- **LDSD** (Passant, 2010). It is a relatedness measure to compute the distance between pairs of resources in Linked Data, which takes into account direct and indirect links, both incoming and outcoming.

Evaluation protocol. For the Ranking SVM a 5-cross fold validation has been used to avoid overfitting. In order to keep results consistent, also the Borda count and the other algorithms have been evaluated on the 5 test sets

fold	*RankSVM*	*Borda*	*Sum*	*DBpediaRanker*	*LDSD*
1	0.3070	0.2502	0.2416	0.2252	0.2343
2	0.3496	0.2304	0.2284	0.2181	0.1853
3	0.3419	0.2892	0.2543	0.2393	0.2194
4	0.3164	0.2520	0.2403	0.2374	0.2312
5	0.3421	0.2396	0.2229	0.2102	0.1899
AVG	0.3314	0.2523	0.2375	0.2260	0.2120
fold	*sa*	*Random*	web^G	web^B	web^D
1	0.1774	-0.0184	0.1693	0.0927	0.2187
2	0.1592	-0.0096	0.1510	0.1550	0.1317
3	0.1900	0.0029	0.2220	0.1693	0.1672
4	0.1416	-0.0354	0.1873	0.1374	0.2468
5	0.2091	0.0238	0.1696	0.1694	0.0702
AVG	0.1755	-0.0073	0.1798	0.1448	0.1669

Table 5: Values of the Spearman's rank correlation coefficient in the experimental evaluation.

derived from the cross fold validation. Since the different features have different ranges in a pre-processing step we normalized them to have zero mean and unit variance. To evaluate and compare the algorithms we used Spearman's rank correlation coefficient (Kendall, 1948) ρ:

$$\rho \quad = \quad 1 - \frac{6 \cdot \sum_i (x_i - y_i)^2}{n(n^2 - 1)}$$

which is a non-parametric measure of statistical dependence between two variables x and y. When x and y are monotonically related, the Spearman correlation coefficient is 1 (or -1). On the contrary, a Spearman correlation of zero means that there is no tendency for y to either increase or decrease when x increases, hence the two variables are completely uncorrelated. In our scenario these two variables represent two possible ranks for a given query s_q.

For each IT resource s_q in the test set we compute the Spearman correlation coefficient between its `ITJobsWatch` ranking list and each of the lists obtained from the algorithms under test. The values represented in Table 5 are the Spearman correlation coefficients averaged among all the resources in the test set. On each row we report the results for the 5 test sets derived from the 5-cross fold validation and the average among the 5 sets.

As expected the *Random* predictor manifests a Spearman correlation near to 0. *Ranking SVM* achieves the best results ($\rho = 0.3314$). It is also interesting to note that using the same ranking features, *Ranking SVM* performs on the average 0.093 better than *Sum* and 0.079 better than *Borda count*. Indeed, differently from the other algorithms, it is designed to optimize a ranking loss function defined on misclassification pairs. It is also noteworthy that when the ranking features are not used together, the results get worse. For example, *LDSD* that does not have any Web-based feature is 0.119 lower than *RankSVM* and 0.04 lower than *Borda count*. Similarly, graph-based (*sa*) or web-based

(web^{σ}) features when used alone are below the results achieved by grouping them together. The differences between the various correlation coefficients expressed above are statistically significant ($p < 0.001$) according to the paired t-test. We also applied the Bonferroni correction (Abdi, 2007) since we performed multiple comparisons. The results show that a supervised approach like *Ranking SVM* is the best solution to aggregate the various features or equivalently combining the rankings. However, also *Borda count*, even if not data-driven, outperforms the other approaches that do not leverage all the available ranking features. This validates the idea of using different and heterogeneous ranking features and information sources to mitigate the weakness of each individual features.

5. Related work

Many techniques have been proposed to automatically evaluate the semantic relatedness between words, texts or concepts in a way corresponding closely to that of human subjects. Most of traditional methods to compute semantic measures exploit particular lexical sources as dictionaries and corpus or well structured taxonomies such as WordNet (Miller, 1995). (Rada et al., 1989) proposes a method for computing semantic relatedness considering the number of edges between two concepts in a semantic network. Specifically, they applied it to MeSH, a term hierarchy for indexing articles in Medline. Following this idea, in (Hirst and St Onge, 1998) the authors applied a similar strategy to WordNet. (Resnik, 1995) introduces a measure of relatedness for WordNet based on the notion of information content, that is a value that is assigned to each concept in a hierachy based on corpus based evidence information. Then, the relatedness between two concepts is determined by the information content of the lowest common subsumer of the two concepts in the hierarchy. The main problem with those methods based on lexical taxonomies such as WordNet is their limited coverage: for example, as reported in (Strube and Ponzetto, 2006), WordNet 2.1 does not include many named entities or many specialized terms, as instead it is needed in the IT domain. The usage of `DBpedia` and more generally of LOD datasets, helps to overcome this problem as the information available within the dataset covers many domains and at the same time receives periodical updates. This can be considered as a point in favour of the approach we present here. Referring to ontologies, in (Mazuel and Sabouret, 2008) the authors present a semantic relatedness measure on ontologies based on the analysis of object properties between concepts. The basic idea behind their method is that the concepts can be connected following different kinds of relations. They first remove the paths semantically incorrect using specific rules and then assign a weight to each path depending on various contextual information such as the type of relations. Compared to our approach, in (Mazuel and Sabouret, 2008) the designer has to assign and semi-automatically check the values assigned to the paths. In our case, we do not need to manually select any relevant path or feature to build the relatedness graph. In (Smyth, 2007) an approach which takes into account individual features of concepts is presented where each particular feature has its own similarity function. Then, final similarity is obtained

as linear combination of the different features where each one is given a different weight depending on its importance. Although the proposal in (Smyth, 2007) shares many element with our approach, the authors consider only concepts within the ontology. Hence, they only work at the schema level. Moreover, they only consider "semantic" features and do not exploit any other statistical and textual information as in our case. A completely different approach based on statistical analysis, namely Latent Semantic Analysis is presented in (Deerwester et al., 1990). LSA is a well known technique used in Information Retrieval that leverages word co-ocurrence from large corpus of textual documents. The main limitation of this technique is that its coverage is restricted to the corpus used as input. More recently other works have focused on using encyclopedic data sources such as Wikipedia to improve coverage with respect to traditional methods based on limited vocabularies. Particularly in (Strube and Ponzetto, 2006) the authors present classic measures adapted to use Wikipedia instead of WordNet as knowledge source. Another work based on Wikipedia is found in (Gabrilovich and Markovitch, 2007). They propose a method based on machine learning techniques to represent words or texts as weighted vectors of Wikipedia-based concepts. The reported results show higher correlation with human judgment than (Strube and Ponzetto, 2006). Our *abstract* feature has been inspired by these works. The main difference is that we simply rely on word occurrences and, for this feature, we do not adopt any machine learning tool. In this way we avoid computational demanding training task while building the relatedness graph in the preliminary phases. Despite the benefits deriving from the usage of Wikipedia as background knowledge source such as its multi-domain and multi-lingual nature, as explained by (Gracia and Mena, 2008), Wikipedia is not able to take into account implicit relationships. In particular they propose to use the Web as knowledge source. They say that many terms do not appear together in any Wikipedia page, but most of them can co-occurr in several web pages, so their implicit relationship could be inferred by accessing the Web. In (Bollegala and Matsuo, 2007) a similarity measure combining page counts information with scores obtained from the analysis of lexico-syntactic patterns extracted from text snippets is proposed. Other web-based semantic similarity measures can be found in (Chen et al., 2006) and (Sahami and Heilman, 2006). They also explore text snippets, but without taking into account page counts as done by (Bollegala and Matsuo, 2007). This is very similar to what we do with web^{σ} features. Differently from (Bollegala and Matsuo, 2007) we do not consider only the results returned by Web search engines but also the information encoded in social data as the one exposed by *Delicious*. Moreover, in this case only Web-based metrics are adopted without considering text-based and graph-based ones. Information theoretic approaches to compute similarities among terms have been shown to be quite effective by computing the information content (IC) of concepts from the knowledge provided by ontologies. These approaches, however, suffer by the coverage offered by the single input ontology. In (Sánchez and Batet, 2013), the authors propose an extension to IC-based similarity measures by considering multiple ontologies in an integrated way. Several strategies are proposed according to which ontol-

ogy the evaluated terms belong to. In (Sánchez et al., 2012) the authors present an ontology-based measure relying on the exploitation of taxonomical features to compute similarity among words. The problem associated to the lack of meaning in tag-based systems is addressed in (Uddin et al., 2013) where a method for finding semantic relationships among tags is proposed. The authors consider not only the pairwise relationships between tags, resources, and users, but also the mutual relationships among them. In (Lau et al., 2009) an ontology-based similarity measurement is presented to retrieve the similar sub-problems that overcomes the problems of synonymity in problem-based case reasoning during the case retrieval. Semantic similarity and relatedness measures between ontology concepts are useful in many research areas. While similarity only considers subsumption relations to assess how two objects are alike, relatedness takes into account a broader range of relations (e.g., part-of). Computing semantic similarity between ontology concepts is an important issue since having many applications in different contexts including: Information Retrieval, ontology matching, semantic query routing, just to cite a few. In (Pirró and Euzenat, 2010), the authors present a framework, which maps the feature-based model of similarity into the information theoretic domain. All these ontological methods, although effective in many cases, require to have a well designed and implemented ontology. While this is the case of domains such as biology, genetics, etc., there are many domains where the ontologies are not very rich and well structured and often they just result in a shallow taxonomy.

Referring to the recent LOD initiative, (Passant, 2010) proposes a measure of semantic distance on Linked Data to identify relatedness between resources. The author takes into account both direct and indirect connections between pairs of resources. In the evaluation section we re-implemented this algorithm and used it for comparison with our approach (see Section 4). Based on the primary SimRank (Jeh and Widom, 2002), the authors in (Olsson et al., 2011) take the first steps to embed semantic relationships in similarity computing. The proposed similarity measure, named SSDM (Structural Semantic Distance Measure), aims at tailoring the primary SimRank to the RDF data format. As its name suggests, the measure leverages both structural and semantic information of a heterogeneous graph to compute similarity. This makes the measure well suited for Linked Data. By arguing that employing distance to compute similarity does not adequately consider all details, the author in (Leal et al., 2012) introduces proximity: the level that indicates how semantically close two resources are according to the weight of the predicates on the path connecting them. The measure considers the weight of edges between nodes, a mapping is defined to convert a link type into an integer value. More recently in (Leal and Costa, 2015), the authors propose to apply deep learning techniques (Schmidhuber, 2015) to compute *proximity* values between `RDF` resources. The proposed approach builds on top of (Leal, 2013) where the implementation and evaluation of the algorithm was presented. An analogous approach is presented in (Yu et al., 2014) where the authors propose an algorithm to compute recommendation in heterogeneous graphs. The authors in (Vocht et al., 2013) make use of available computation resources to find paths in structured data. Applying

these algorithms to Linked Data can facilitate the resolving of complex queries that involve the semantics of the relations between resources. An approach is introduced for finding paths in Linked Data that takes into account the meaning of the connections and also deals with scalability. An efficient technique combining pre-processing and indexing of datasets is used for finding paths between two resources in large datasets within a couple of seconds. All these semantic approaches base on graph exploration and path identification and counting assume the designer already knows all the nodes/entities within the semantic graph. This is not the case of our approach where we only need to know a set of seed nodes to explore the dataset and build the relatedness graph. This is particularly useful in all those situations where the initial knowledge owned by the designer of the intelligent system is limited to a subset of the actual one. Moreover, the computation of relatedness values (in our case a ranking value) is contextual to the exploration. Finally it is worth noticing that, differently from our approach, a training set is needed in order to compute the relatedness values between nodes. Moreover, a weight needs to be set for each relevant property within the selected dataset.

In (Zadeh and Reformat, 2012) a metric for calculating semantic similarity between two resources based on features, called FuzzySim, is presented. Properties/predicates are considered as features. Furthermore, it is hypothesized that each group of properties has a specific influence on the overall similarity. As a result, the set of properties in an RDF graph is intuitively classified into subsets according to their level of importance. The similarity between two resources with regard to all the subsets of properties in the RDF graph is calculated by combining all sub-metrics using a fuzzy function whose aim is to incorporate the human judgment of similarity. As for the metrics described previously, FuzzySim needs to manually assign a value of importance to the (set of) relevant properties in order to classify their importance. This is not the case of our metric that does not need any manual pre-processing step.

Compared to the approaches proposes in the literature on semantic relatedness we may summarize the following points in favour of the approach we present in this paper:

- The use of `DBpedia` makes our approach applicable to other knowledge domains. Moreover, even though we exploit the semantics of `RDF` datasets we do not only consider the ontological information they encode to compute relatedness between nodes. This allows our approach ot be used also for those domain where the underlying ontologyis not very rich or structured.

- We do not rely on a single metric but we combine more than one in order to exploit the advantages of each of them. Indeed, we adopt an ensemble of graph-based, Web-based and text-based metrics.

- As for the graph-based metric, we do not need to know all the elements a priori. We just need an initial set of seed nodes resulting meaningful to represent the knowledge domain. Starting from them, the spreading activation algorithm is able to discover new nodes belonging to the knowledge

domain which were initially unknown .

- We do not necessarily need any initial dataset already containing relatedness measures to train our model. All the metrics we propose here do not require a training step. As for ranking aggregation, the unsupervised approach for features aggregation based on Borda count is able to compute relatedness values without the need of a training set.

6. Conclusion and future work

In this paper we have presented an approach to rank LOD resources with a particular emphasis on the IT domain. The measure we propose makes use of several ranking features to compute the relatedness between pairs of resources. In particular, both graph-based, text-based and web-based features have been exploited to provide effective results. Thanks to the usage of Linked Open Data datasets, such as `DBpedia`, our approach is able to capture semantic relations between resources beyond the possibilities offered by traditional text-based approaches. We have employed this measure to build a labelled graph of related concepts. In particular, in this work we focused on concepts belonging to the IT domain (programming languages, databases, technologies, frameworks, etc.).

The produced graph may represent a key component in the design and development of expert systems whose goal is to support the user in selecting items resulting relevant and related to the ones they might be interested in. In particular, for the IT domain, an expert systems might guide the user through the decision-making process of finding technologies and tools which can be associated, to some extent, to those needed for a specific task. As a way of practical applications for the proposed approach we may cite at least: *expert finding*, *task-driven exploratory search for learning*, *software components selection in complex system design and development*. In expert finding scenarios, the main goal is to find experts able to cope with a specific task. There are many situation where an exact matching between the requested skills and the available ones is not possible. We may think at the allocation of employees within a company to solve a task for which specific skills are needed. By looking at the CV of available employees, in case of non-exact match, the company might be interested in those whose skills are as much related as possible to the requested ones. As for the *task-driven exploratory search for learning* use case we may develop an expert system able to guide the user in a learning process whose aim is to improve their knowledge on specific technologies and tools needed to solve a task. Suppose the user is asked to learn PHP to develop a Web portal. Hence, the expert system may suggest the user to take a look also to MySQL as it is the most used DBMS when developing PHP projects. The system may also allow the users to explore related technologies thus making them aware of new tools that could result useful for the development of the project. Finally, also *software components selection in complex system design and development* may benefit from the availability of an expert system using our relatedness graph in the IT domain. Indeed, while looking for a specific component, library or

API, the system is able to suggest to the user other related components that may contribute to the final design of a complex system. In case the project uses Django as the framework for Python development, the expert system could suggest to adopt Aptana Studio as Integrated Development Environment.

Although the presented approach is mainly focused on the IT domain, it can be extended to other knowledge domains thus allowing the creation of a semantic-aware class of expert systems. Imagine the domain of cultural heritage in a touristic application where the system suggests new sites, museums, attractions which are similar to the ones already visited by the user. Although it is very general, adapting the whole framework to a new domain may present some limitation due to some factors. First of all, the selection of seed nodes may not be as straight as for the IT domain. In this paper we exploit the information available in `ITJobsWatch` and map them to `DBpedia`. This process presents two main critical points: (i) the mapping may not be a simple and direct process. Automatic entity linking (Gangemi, 2013; Shen et al., 2015) may introduce some noise in the data as it may not be correct for all the items; (ii) the selection of seed nodes cannot be done automatically but left to one or more domain experts. This is the case for domains where we do not have an information source analogous to `ITJobsWatch`. Another factor that might hinder the application of our approach to other knowledge domains is the low quality of the selected `RDF` data. The accuracy and richness of the relatedness graph reflects that of the original `RDF` data. In case we have a poorly connected `RDF` graph it may result difficult to discover relatedness connections among entities. Actually, this is not the case of the dataset as a whole but also of `RDF` subgraphs in `DBpedia` representing the domain of interest. As `DBpedia` reflects the structure of Wikipedia pages, we may find knowledge domains which are very well described and connected while other ones which contains a few elements and associated information. Finally, if the textual description coming with `RDF` resources does not contain relevant information, the text-based feature we adopt in our approach may not contribute very well to the final aggregate relatedness score.

For a better understanding on how to overcome the above limitations as future work we are interested in testing our relatedness measure on different domains in order to evaluate how well the approach can be generalized and exploiting the knowledge encoded in further LOD datasets in addition `DBpedia`. This paves the way to a new research challenge which has been only partially investigated by the scientific community: how to estimate the quality of LOD datasets. In fact, while a lot of literature can be found describing metrics to evaluate data quality applied to the relational model (Kritikos et al., 2013), almost nothing is available when the problem refers to `RDF` data. We believe that metrics for this latter kind of data need to be investigated also in an application-oriented perspective. Depending on the application built on top of `RDF` datasets, we need to investigate different metrics such as *knowledge coverage*, connection of the underlying graph and ontological structure of the data.

Among future research directions exploiting the results presented here, we also see the design of user interfaces able to exploit the relatedness graph to-

gether with the semantics associated to nodes. Indeed, if the user is interested in the configuration of a complex system, the intelligent application must be able to guide the user by navigating through related resources and, at the same time, show nodes belonging to specific classes or categories that could be of interest for the user.

As stated in Section 1 the notion of relatedness does not refer to a single class of objects but to multiple classes. For this reason the application of our relatedness graph could feed the recent research on cross-domain recommender systems (RS) (Cantador and Cremonesi, 2014). While historically recommendation engines were designed to work in a specific domain (movies, books, music), recently there has been a growing interest in expert systems able to recommend items belonging to domains different from the one where the user profile belongs to. For example, we recommend books to users given their profile in the movie domain.

Another future research direction for the results we present here is how to integrate social information for community discovery applications. As an example we may consider the academic scenario where, given a set of authors, we have not only information on their papers, their topics and the conference/journals they have been published in but also social information about co-authorships, authors who published in the same conferences and journal and so on. All this information can be combined to allow an intelligent system to identify communities/clusterings of people working on related topics but above all in related disciplines.

Acknowledgments. The authors acknowledge partial support of HP IRP 2012 - Grant CW267313 and PON02_00563_3470993 VINCENTE.

References

H. Abdi. *Bonferroni and Sidak corrections for multiple comparisons.* Sage, Thousand Oaks, CA, 2007.

Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3): 1–22, 2009a.

Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. Dbpedia - A crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009b.

Danushka Bollegala and Mitsuru Matsuo, Yutaka. Measuring semantic similarity between words using web search engines. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 757–766, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7.

Iván Cantador and Paolo Cremonesi. Tutorial on cross-domain recommender systems. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 401–402, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2668-1.

Hsin-Hsi Chen, Ming-Shun Lin, and Yu-Chuan Wei. Novel association measures using web search with double checking. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 1009–1016, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11:453–482, 1997.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 613–622, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0.

Frank Manola, Eric Miller. RDF Primer. `http://www.w3.org/TR/rdf-primer`, 2004.

Norbert Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Trans. Inf. Syst.*, 7(3):183–204, July 1989. ISSN 1046-8188.

Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

Aldo Gangemi. A comparison of knowledge extraction tools for the semantic web. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 351–366. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38287-1.

Jorge Gracia and Eduardo Mena. Web-based measure of semantic relatedness. In *In Proc. of 9th International Conference on Web Information Systems Engineering (WISE 2008), Auckland (New Zealand*, pages 136–150. Springer, 2008.

G. Hirst and D. St Onge. *Lexical Chains as representation of context for the detection and correction malapropisms.* The MIT Press, May 1998.

Glen Jeh and Jennifer Widom. Simrank: A measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 538–543, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X.

Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5.

Maurice G. Kendall. *Rank correlation methods.* Griffin, London, 1948.

Kyriakos Kritikos, Barbara Pernici, Pierluigi Plebani, Cinzia Cappiello, Marco Comuzzi, Salima Benbernou, Ivona Brandic, Attila Kertész, Michael Parkin, and Manuel Carro. A survey on service quality description. *ACM Comput. Surv.*, 46(1):1, 2013.

Adela S. M. Lau, Eric Tsui, and W. B. Lee. An ontology-based similarity measurement for problem-based case reasoning. *Expert Syst. Appl.*, 36(3): 6574–6579, 2009.

José Paulo Leal. Using proximity to compute semantic relatedness in RDF graphs. *Comput. Sci. Inf. Syst.*, 10(4):1727–1746, 2013.

José Paulo Leal and Teresa Costa. Tuning a semantic relatedness algorithm using a multiscale approach. *Comput. Sci. Inf. Syst.*, 12(2):635–654, 2015.

José Paulo Leal, Vânia Rodrigues, and Ricardo Queirós. Computing semantic relatedness using dbpedia. In *1st Symposium on Languages, Applications and Technologies, SLATE 2012, Braga, Portugal, June 21-22, 2012*, pages 133–147, 2012.

Tie-Yan Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, March 2009. ISSN 1554-0669.

Gary Marchionini. Exploratory search: from finding to understanding. *Commun. ACM*, 49(4):41–46, 2006. ISSN 0001-0782.

Laurent Mazuel and Nicolas Sabouret. Semantic relatedness measure using object properties in an ontology. In *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08, pages 681–694, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-88563-4.

George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38 (11):39–41, November 1995. ISSN 0001-0782.

Roberto Mirizzi, Azzurra Ragone, Tommaso Di Noia, and Eugenio Di Sciascio. Ranking the linked data: the case of dbpedia. In *10th International Conference on Web Engineering (ICWE 2010)*, 2010.

Rabia Nuray and Fazli Can. Automatic ranking of information retrieval systems using data fusion. *Inf. Process. Manage.*, 42(3):595–614, May 2006. ISSN 0306-4573.

Catherine Olsson, Plamen Petrov, Jeff Sherman, and Andrew Perez-Lopez. Finding and explaining similarities in linked data. In *Proceedings of the Sixth International Conference on Semantic Technologies for Intelligence, Defense, and Security, Fairfax, VA, USA, November 16-17, 2011*, pages 52–59, 2011.

Alexandre Passant. Measuring semantic distance on linking data and using it for resources recommendations. In *Proceedings of the AAAI Spring Symposium "Linked Data Meets Artificial Intelligence"*, 3 2010.

Giuseppe Pirró and Jérôme Euzenat. A feature and information theoretic framework for semantic similarity and relatedness. In *Proceedings of the 9th International Semantic Web Conference on The Semantic Web - Volume Part I*, ISWC'10, pages 615–630, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-17745-X, 978-3-642-17745-3.

Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query`, 2008.

Ross Quillian. Semantic memory. In *Semantic Information Processing*, pages 216–270. MIT Press, 1968.

Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1):17–30, 1989.

Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 448–453, 1995.

Philip Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res. (JAIR)*, 11:95–130, 1999.

S.M. Ross. *A first course in probability*. Macmillan, 1976. ISBN 9780029796207.

Mehran Sahami and Timothy D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *Proceedings of the 15th international conference on World Wide Web*, WWW '06, pages 377–386, New York, NY, USA, 2006. ACM. ISBN 1-59593-323-9.

David Sánchez and Montserrat Batet. A semantic similarity method based on information content exploiting multiple ontologies. *Expert Syst. Appl.*, 40(4): 1393–1399, 2013.

David Sánchez, Montserrat Batet, David Isern, and Aïda Valls. Ontology-based semantic similarity: A new feature-based approach. *Expert Syst. Appl.*, 39 (9):7718–7728, 2012.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

Bernhard Schlkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Transaction on Signal Processing*, 45: 2758–2765, 1997.

Wei Shen, Jianyong Wang, and Jiawei Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *Knowledge and Data Engineering, IEEE Transactions on*, 27(2):443–460, Feb 2015. ISSN 1041-4347.

Barry Smyth. Case-based recommendation. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, pages 342–376. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72078-2.

Michael Strube and Simone Paolo Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, AAAI'06, pages 1419–1424. AAAI Press, 2006. ISBN 978-1-57735-281-5.

Mohammed Nazim Uddin, Trong Hai Duong, Ngoc Thanh Nguyen, Xin-Min Qi, and GeunSik Jo. Semantic similarity measures for enhancing information retrieval in folksonomies. *Expert Syst. Appl.*, 40(5):1645–1653, 2013.

Laurens De Vocht, Sam Coppens, Ruben Verborgh, Miel Vander Sande, Erik Mannens, and Rik Van de Walle. Discovering meaningful connections between resources in the web of data. In *Proceedings of the WWW2013 Workshop on Linked Data on the Web, Rio de Janeiro, Brazil, 14 May, 2013*, 2013.

Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. Personalized entity recommendation: A heterogeneous information network approach. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 283–292, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2351-2.

Parisa D Hossein Zadeh and Marek Z Reformat. Fuzzy semantic similarity in linked data using the owa operator. In *Fuzzy Information Processing Society (NAFIPS), 2012 Annual Meeting of the North American*, pages 1–6. IEEE, 2012.

Appendix A. Background technologies: RDF and SPARQL

In this section we briefly recap the basic notions of two Semantic Web technologies used in our approach: RDF (Eric Miller, 2004) and SPARQL (Prud'hommeaux and Seaborne, 2008).

Resource Description Framework - RDF.
The Resource Description Framework (RDF) is a general model for describing information about resources on the Web. It has been developed by the World Wide Web Consortium (W3C) in 1998 as the building block for the Semantic Web. It allows to represent Web entities and their relations as well as to attach to them a machine understandable and processable meaning (semantics) that can be further exploited to perform automatic reasoning tasks able to infer new knowledge from the explicitly stated one. Thanks to RDF, resources are made available on the Web, enabling applications to exploit them by taking into account their meaning. Each statement about resources is modeled in the form of a triple: *subject-predicate-object*. *Subjects* and *predicates* are represented by URIs, while *objects* can be identified either by URIs or by literals (data values). As an example, the two following triples are valid RDF statements about the object-oriented programming language *Java*:

```
<http://dbpedia.org/resource/Java_(programming_language)>
   <http://www.w3.org/2000/01/rdf-schema#label>
      "Java"@en

<http://dbpedia.org/resource/Java_(programming_language)>
   <http://dbpedia.org/ontology/influenced>
      <http://dbpedia.org/resource/PHP>
```

where we state that we may refer to the the Java programming language with the word "Java" in English and that it influenced the development of PHP. RDF information representation can be formally modeled through a labeled directed graph. In fact, if we consider all the RDF statements (triples) as a whole, what we get is a graph, where nodes are resources connected to other resources or to literal values through predicates (the edges of the graph).
RDF can be serialized by means of different syntaxes. The most compact is the so called turtle syntax that allows us to use prefixes to shorten the URIs. The turtle version of the two triples above is:

```
@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix dcterms: <http://purl.org/dc/terms/>
@prefix category: <http://dbpedia.org/resource/Category:>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

dbpedia:Java_(programming_language)
   rdfs:label  "Java"@en ;
   dbpedia-owl:influenced   dbpedia:PHP .
```

From an ontological point of view, an interesting built-in RDF predicate is http://www.w3.org/1999/02/22-rdf-syntax-ns#type. It states that a resource is an instance of a class.

```
@prefix dbpedia: <http://dbpedia.org/resource/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>

dbpedia:Java_(programming_language)
   rdf:type   dbpedia-owl:ProgrammingLanguage .
```

The previous triple asserts that *Java* is an instance of the class *Programming Language*.

Simple Protocol and RDF Query Language - SPARQL.
SPARQL is the de facto query language for RDF datasets. The language reflects the graph-based nature of the underlying data model. Indeed, graph-matching is its query mechanism. A basic SPARQL query is of the form:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?c ?l
WHERE {
   <http://dbpedia.org/resource/Java_(programming_language)>
       dcterms:subject  ?c .
   ?c  rdfs:label ?l.
}
```

where we ask for all possible values that can be assigned to the variables ?l and ?c in order to match the graph pattern expressed in the WHERE clause.