Rank Prediction for Semantically Annotated Resources

Pasquale Minervini, Nicola Fanizzi, Claudia d'Amato, Floriana Esposito
Dipartimento di Informatica, LACAM Laboratory
Università degli Studi di Bari "Aldo Moro"
Via E. Orabona, 4 - 70125 Bari - Italy
{ pasquale.minervini, nicola.fanizzi, claudia.damato, floriana.esposito } @uniba.it

ABSTRACT

In the context of semantic knowledge bases, we tackle the problem of ranking resources w.r.t. some criterion. The proposed solution is a method for learning functions that can approximately predict the correct ranking. Differently from other related methods proposed, that assume the ranking criteria to be explicitly expressed (e.g. as a query or a function), our approach is data-driven, being able to produce a predictor detecting the implicit underlying criteria from assertions regarding the resources in the knowledge base. The usage of specific kernel functions encoding the similarity between individuals in the context of knowledge bases allows the application of the method to ontologies in the standard representations for the Semantic Web. The method is based on a kernelized version of the Perceptron Ranking algorithm which is suitable for batch but also online problem settings. Moreover, differently from other approaches based on regression, the method takes advantage from the underlying ordering on the ranking labels. The reported empirical evaluation proves the effectiveness of the method at the task of predicting the rankings of single users in the Linked User Feedback dataset, by integrating knowledge from the Linked Open Data cloud during the learning process.

1. INTRODUCTION AND MOTIVATION

Ranking a set of individual objects, as the result of a relation sought between them and their relative ranks, is a fundamental task in many research fields with widespread applications. In a typical scenario, a user may need to rank resources (e.g. documents) returned as a result of *search/query answering/retrieval* (from a corpus, a database, etc.). An example of ranking related to relevance-feedback mechanisms is the task known as *collaborative filtering* which aims to detect the relevance of information items based on rankings previously acquired from a community of users. This ranking cannot depend entirely on a notion of relevance expressed in terms of the knowledge base semantics, but it may include users *in the loop* as providers of examples of ranked resources according to their own relevance criteria (especially when these would be hard to formally express). In this way a sort of group intelligence can emerge.

There are a plethora of important Web applications that require solutions for this general problem. For example, one such application may want to associate every client of an online shop (for songs, movies, books, etc.) with a ranking on the products, depending on a) previous rankings of the client on other products (along with some background knowledge on the relationships among the products), or b) on previous rankings of other clients on the same product (along with background knowledge on the relationships among the clients), or c) on both options. Such a ranking may be useful for showing a client the top-ranked products as buying suggestions when the client is browsing the online shop. In a similar way, the members of a social network may be associated with a ranking on, e.g., their interests.

Solving ranking problems may turn out to be a hard task, since it is generally difficult to define a general and precise measure of the relevance when this criterion cannot be expressed with general logical axioms but rather in the form of multiple relevance orders, preferences, etc.. Still it may be possible to (partially) elicit the required criterion by induction, exploiting imprecise information that can often be expressed by means of examples of rankings for a limited number of resources.

In general, *learning to rank* is more difficult than other related tasks such as *learning to classify*. The problem can be cast in the framework of learning from examples. Given previously ranked instances, the aim is to learn a function which can be used to assign a meaningful rank to new incoming unranked instances. Essentially, previous (partial) pointwise indications of the intended relevance of some resources (e.g. ratings, feedback from users) w.r.t. some criterion have to be exploited. Ranking methodologies proposed in the context of *Information Retrieval* from textual corpora settings cannot be transposed straightforwardly to the *Semantic Web* context that leverages formal descriptions of the available resources (e.g. see [7]). Their logic-oriented nature requires specific approaches to the problem that can comply with the underlying semantics of the data.

While exploiting the structure of the descriptions (e.g. see [1]) represents a partial solution to the problem which cannot fully exploit the richness of the underlying logic representation, purely logical methodologies investigated in a few related works may fall short in terms of scalability. For example, the problem of semantic *matchmaking* is tackled by means of non-monotonic inferences in Description Logics (henceforth DLs) such as concept *abduction* and *contraction* [9]. Further drawbacks come from the language-dependence of some of the required operations and the complexity of the related inference services with the growing expressiveness of the DL language. Other approaches tackle the problem by explicitly representing the preferences in the knowledge bases with conditional statements for ranking objects in ontologies [15], or, conversely, recur to fuzzy extensions of the standard DL languages [16] to offer alternate ways for finding the best (top k) answers to queries [18].

Statistical learning has been shown to provide valid techniques to produce alternative models that enable forms of inductive classification in DLs [5, 8, 3, 10]. Also *ranking* can be performed inductively, casting the problem as learning a suitable function from a training sample of ranked data. Decomposing the problem into multiple class learning problems would not exploit efficiently the inherent ordering in the ranks and it may require a further level of approximation in the combination of multiple classifiers. Ideally, even the number of ranks need not be specified, though typically the training data comes with a relative ordering, specified by the assignment to one of an ordered sequence of labels.

We propose a method based on kernels that is able to learn ranking functions from examples of ranked individuals w.r.t. some (potentially unknown) target criterion in terms of a DL knowledge base. This function can be then used to predict a correct rank (or a close approximation) for newly acquired individuals. The algorithm is computationally efficient and can work in an online fashion, so that it may improve the quality of the ranking function as long as new ranked individuals are made available. The paper presents also the outcomes of a thorough experimental evaluation with real ontologies where the rankings used for the training phase are determined using suitable automated methods. This evaluation proves the effectiveness of the method in terms of a standard measure for this task (*average loss*).

In this paper, the basics of semantic knowledge bases represented in DL are recalled (see next section). Then, after presenting the essentials of the kernel methods in Sect. 3 and a specific family of DL-kernels, a definition of the ranking problems is formalized and a kernel-based method for learning ranking functions is described in Sect. 4. Given its implementation, we can also present in Sect. 5 an experiment on a real-world ontological knowledge base.

2. PRELIMINARIES

For brevity, the basics of the representation utilized are informally recalled with no claim to exhaustiveness (see [2] for a thorough reference).

The building blocks are in a *vocabulary* $\langle N_C, N_R, N_I \rangle$ made up, respectively, of a set of *concept* names (*classes*), a set of *role* names (*properties*) and a set of *individual* names (*resources*). A DL language provides specific constructors and rules for expressing complex descriptions when applied to the symbols in the vocabulary with a particular syntax and semantics that has a correspondence with First-Order Logic (and its extensions).

A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ will be considered as composed by a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$. $\mathcal{T}$ is a set of axioms $\alpha \sqsubseteq \delta$, where[1] $\alpha = C \in N_C$ (resp. $\alpha = R \in N_R$) is an atomic name and δ is a legal concept (role) description constructed on concept and role names by means of the syntax of the operators allowed by the specific DL language.

A peculiar feature of these representations is the underlying *Open World Assumption* (OWA) made on their semantics, which is particularly convenient for the Semantic Web. Formally the semantics is defined in terms of interpretations. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ assigns the meaning to these formulæ via $\cdot^{\mathcal{I}}$ mapping the names to the corresponding element based on the domain $\Delta^{\mathcal{I}}$: resp. concepts $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, binary relations $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and individuals $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* (is a model of) an axiom[2] $\alpha \sqsubseteq \delta$ iff $\alpha^{\mathcal{I}} \subseteq \delta^{\mathcal{I}}$. The ABox $\mathcal{A}$ contains assertions (ground axioms) regarding the world state, e.g. $C(a)$ and $R(a,b)$ meaning that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. $\mathsf{Ind}(\mathcal{A})$ will denote the set of all individuals occurring in the ABox. Normally, the interpretations of interest are limited to the models of the knowledge base, i.e. those that satisfy each axiom (assertion) therein.

For the targeted problem we will exploit *instance checking*, that roughly amounts to deciding whether an individual a is an instance of a concept C [2]: $\mathcal{K} \models C(a)$. The inherent incompleteness of the knowledge bases under an open-world semantics may cause the impossibility to ascertain the class-membership w.r.t. a query concept and to its negation through instance-checking. The related *retrieval* service may be exploited to find the sets of individuals occurring in the ABox that can be proven to belong to a given concept.

3. LEARNING WITH KERNELS IN DL

A learning task requires finding an inductive model which can be adopted by a decision procedure to predict, given an input instance, the correct value for the function to be approximated both efficiently (evaluating a simple model) and effectively (close approximation). Kernel methods [17] are particularly well suited because the learning algorithm (*inductive bias*) and the choices of the kernel function (*language bias*) are almost completely independent. Discovering linear relations has a long tradition of research, with algorithms that are both efficient and well understood. A computational shortcut makes it possible to represent linear models efficiently in high-dimensional spaces to ensure adequate representational power: a kernel function. Different kernel functions may be related to different hypothesis spaces. Hence, the same kernel machine can be applied to different representations, provided that suitable kernel functions are available.

Kernel methods are characterized by two aspects:

- data items are embedded (implicitly) into a vector space, the *feature* (*embedding*) space; this depends on the specific data types and domain knowledge expected for the particular data source;
- linear models are sought among the images of the data items in the feature space; the algorithms are implemented in such a way that only the inner products of the embedded points are needed and the products can be computed efficiently directly from the original data items using a *kernel* function.

[1]Even more complex inclusion axioms are generally also admitted, where α is not bound to be atomic.

[2]Definitions, denoted $\alpha \equiv \delta$, are axioms that are satisfied by some $\mathcal{I}$ when $\alpha^{\mathcal{I}} = \delta^{\mathcal{I}}$ holds.

The *learning* component is general-purpose, robust and also efficient, requiring an amount of computational resources that is polynomial in the size and number of data items even when the dimension of the embedding space grows exponentially. An algorithm may be adapted to structured spaces (e.g. trees, graphs [17]) by merely replacing the kernel function with a suitable one. Examples of the target concepts are to be provided to the learning algorithm to produce a definition for the target function in the form of a linear decision function depending on a tuple of weights.

3.1 Kernel Methods Essentials

Many learning problems can be reduced to the approximation of linear functions. In a simplified setting, consider an *input* space $\mathcal{X}$ of training instances (e.g. represented by binary or real tuples) extended with an additional feature $y \in Y$ indicating an implicit target value: $(x, y) \in \mathcal{X} \times Y$ (these are named *examples*). This target value may represent the membership w.r.t. a class (*classification* problem) or the value of a function to be predicted (*regression* or *ranking* problem). We will consider learning from a *sample* $S \subseteq (\mathcal{X} \times Y)$ of N examples.

A prototypical algorithm is the Perceptron, a procedure for learning the coefficients of linear classifiers (in a classic setting where $Y = \{-1, +1\}$ contains membership values). For each incoming training example (x_i, y_i), the algorithm predicts a value according to the decision function $g(h_w(x_i); w)$ based on the linear model h_w determined by the current choice of weights w. The algorithm compares the outcome $\hat{y}_i = g(h_w(x_i); w)$ with the correct label y_i which is known for the examples considered during the training phase. On erroneous predictions, the weights w are revised depending on the examples that provoked a mistake.

However, simple and efficient methods for this setting apply only to linearly separable problems. This issue may often be circumvented by resorting to the *kernel trick*, that essentially consists in finding a mapping ϕ of the instances onto a suitable feature space, allowing for the linear separation between examples belonging to different classes. The embedding space has likely many more (even an infinite number of) dimensions. However, the mapping is never explicitly performed; a *kernel* function, corresponding to the inner product of the transformed vectors in the new space, ensures that an embedding exists [17]: $\kappa(x_i, x) = \phi(x_i) \cdot \phi(x)$.

In a kernel method, each training instance x_i is thought as generating a local density function - the kernel $\kappa(\cdot, x_i)$ - of its own that assigns a probability to each point in the space. The kernel function depends on some distance $d(x, x_i)$ from x to the instance x_i. The density estimate as a whole is just the normalized sum of all the local kernel functions: $P(x) = (1/N)\sum_{x_i \in S} \kappa(x, x_i)$, where S is the local set of instances considered. Learning is carried out by considering a weighted combination of all the predictions related to each training example. The weight of the i-th example for a query instance x is given by the value of the kernel $\kappa(x, x_i)$ (weighted vote). Kernel machines are not limited to learning classifiers only but can be used also for solving regression and ranking problems using similar methods [17].

3.2 DL-Kernels

The choice of kernel functions is very important as their computation should be efficient enough for controlling the complexity of the overall learning process. Although some kernels have been proposed for instances expressed in First Order Logic fragments, only a few kernel functions for individuals have been proposed in the literature.

In this work, we resort to a set of general kernels for individuals in DL knowledge bases, stemmed from those defined in [10], that exploit a notion of similarity in the context of a set of concepts:

Definition 3.1 (DL kernels). *Given a knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ *and a set of concept descriptions* $\mathsf{F} = \{F_i\}_{i=1}^m$ *defined in terms of* $\mathcal{K}$*, the* kernel function *based on* F *and* $p \in \mathbb{R}_+$*, is a mapping* $k_p^{\mathsf{F}} \colon \mathsf{Ind}(\mathcal{A}) \times \mathsf{Ind}(\mathcal{A}) \to [0, 1]$ *defined as follows:*

$$\forall a, b \in \mathsf{Ind}(\mathcal{A}) \qquad k_p^{\mathsf{F}}(a, b) = \left[\sum_{i=1}^{m} (\kappa_i(a, b))^p\right]^{1/p}$$

where $\forall i \in \{1, \ldots, m\}$ *the i-th* simple kernel *is defined:*

$$\kappa_i(a, b) = \pi_i^+(a)\pi_i^+(b) + \pi_i^-(a)\pi_i^-(b)$$

with $\pi_i^+(x) \approx \Pr^{\mathcal{K}}(F_i(x))$ *and* $\pi_i^-(x) \approx \Pr^{\mathcal{K}}(\neg F_i(x))$.

As feature concepts F_i, the very set of the (primitive) concepts (possibly weighed with respect to their ability to discern individuals) in $\mathcal{K}$ may be considered. Alternatively, an ad hoc learning algorithm [10] which learns the best concept descriptions with respect to a given discernibility criterion could be employed. For a generic individual x in $\mathcal{K}$, the value of the function $\pi_i^+(x)$ (resp. $\pi_i^-(x)$) approximates the probability value that x is instance of F_i (resp. of $\neg F_i$). Specifically, for a given individual x, if $\mathcal{K} \models F_i(x)$ (resp. $\mathcal{K} \models \neg F_i(x)$) occurs then $\pi_i^+(x) = 1$ and $\pi_i^-(x) = 0$ (resp. $\pi_i^-(x) = 1$ and $\pi_i^+(x) = 0$). In some cases, because of the OWA, the class-membership (and non-membership) of some individual x w.r.t. F_i cannot be ascertained (models can be constructed both for $F_i(x)$ and for $\neg F_i(x)$). Since both possibilities are open, an intermediate value u_i is assigned to reflect such uncertainty. This value could be set equal to 0.5 for expressing total uncertainty or, for estimating the probability that x may belong (not belong) to the extension of F_i ($\neg F_i$), the sizes of the *retrieval* sets of F_i and its complement w.r.t. the current ABox can be exploited. Specifically, $\Pr^{\mathcal{K}}(F_i(x)) = |\mathsf{retrieval}(F_i)|/|\mathsf{Ind}(\mathcal{A})|$ (resp. $\Pr^{\mathcal{K}}(\neg F_i(x)) = |\mathsf{retrieval}(\neg F_i)|/|\mathsf{Ind}(\mathcal{A})|$) where $\mathsf{Ind}(\mathcal{A})$ represents the set of all individuals occurring in $\mathcal{A}$ and *retrieval* is the retrieval inference procedure [2] that assesses the individuals of $\mathcal{K}$ that are instances of F_i. Of course statistical approximations would fall short in case a few individuals were available.

The rationale for these kernels is that similarity between individuals is determined by their similarity w.r.t. each concept in a given committee of features. Two individuals are maximally similar w.r.t. a given concept F_i if they exhibit the same behavior, i.e. both are instances of the concept or of its negation. Conversely, the minimal similarity holds when they belong to opposite concepts.

4. KERNEL-BASED RANKING FOR DL

In general, *learning to rank* may be described as the following task. Given a set of ranked examples (x, y), where each individual object x in some input space $\mathcal{X}$ is assigned with a label y from an ordered set Y, the goal is to predict the rank for new unlabeled instances $(x, \cdot)$.

This could be tackled as a *regression* or *classification* problem by treating the ranks as real-values or the assignment to a particular rank value as a classification.

Formally:

Definition 4.1 (ranking problem). *Let $\mathcal{X}$ be a set of individual objects and Y be a set of ranks, endowed with a total order relation $\leq_Y$, such that one of the following cases is considered:*

- $|Y| = r \in \mathbb{N}$ *(discrete case)*
- $\exists$ *bijection* $\beta : Y \to \mathbb{R}^+$ *(continuous case)*

A ranking problem *is specified as follows:*

given: *a sample set* $S = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subseteq \mathcal{X} \times Y$, *where* $y_i \in Y$ *is the* rank *of* $x_i \in \mathcal{X}$

find: *a* ranking function $\rho : \mathcal{X} \to Y$ *that induces a relation* $\preceq_\rho$ *over* $\mathcal{X}$ *that is a partial (resp. total) order.*

The simplest way to define the set of ranks in the continuous case is $Y = \mathbb{R}^+$ or, in the other case, $Y = \{1, \ldots, r\}$.

In this setting, we may say that x_i is *preferred* over x_j, denoted $x_i \preceq_\rho x_j$, iff $\rho(x_i) \leq_Y \rho(x_j)$. The objects x_i and x_j are *incomparable* (cannot be ordered) when $y_i =_\rho y_j$. In the categorical (discrete) case, one may consider that the (partial) ordering partitions the input space into r equivalence classes.

A reduction to a classification problem in the discrete case, would not make use of all of the available information; on the other hand the flexibility inherent in the ordering requirement is the price for the reduction to regression. Conversely, in a regression setting a specific metric is required to convert the ranks into real values. This may be difficult in general and makes regression more sensitive to the rank representation rather than to their ordering [12]. It is therefore preferable to treat ranking as a problem in its own right and design specific algorithms able to take advantage of the specific nature of that problem [17]. This can be a by-product of the adoption of specific kernels and the related embedding.

The discrete case setting can be transposed into the problem of predicting the relative ordering of all possible pairs of examples, hence obtaining two-class classification problems. The drawback of this approach would be the extra computational effort required since the sample size for the algorithm grows quadratically with the number of examples. If, on the other hand, the training data is given in the form of all relative orderings, a set of ranks can be generated as equivalence classes of the equality relation with the induced ordering.

Preliminarily, we will assume an implicit kernel-defined feature space with the corresponding feature mapping ϕ so that $\phi(x_i)$ is in $\mathbb{R}^n$ for some n, $1 \leq n \leq \infty$.

A *linear ranking rule* function embeds the input data into the real axis by means of a linear function in the feature space $f(x) = (\vec{w} \cdot \phi(x))$. The real-value can be then converted to a rank by means of an r-dimensional vector of thresholds $\vec{\theta}$ (to be learned) with the θ_y's ordered according to the underlying relation on Y, i.e. $y \leq y'$ implies $\theta_y \leq \theta_{y'}$.

Definition 4.2 (ranking rule). *Given the thresholds $\vec{\theta}$ and a linear function f, the* ranking rule *is defined by:*

$$\forall x \in \mathcal{X} \quad \rho(x; \vec{w}, \vec{\theta}) = \min\{y \in Y \mid f(x) < \theta_y\}$$

or, in a dual representation, given the kernel function κ:

$$\rho(x; \vec{\alpha}, \vec{\theta}) = \min \left\{ y \in Y \mid f(x) = \sum_{i=1}^{N} \alpha_i \kappa(x, x_i) < \theta_y \right\} \quad (1)$$

where $\vec{w} = \sum_{i=1}^{N} \alpha_i \phi(x_i)$.

Algorithm 1 function DL-Rank($TSet, \vec{\alpha}, \vec{\theta}$)

```
Input:
    $TSet = \{(x_t, y_t)\}_{t=1,\ldots,N}$: set of training examples
Output:
    $\vec{\alpha} \in \mathbf{R}^N$: coefficients for the ranking function
    $\vec{\theta} \in \mathbf{R}^r$: ranking thresholds
 1: $\vec{\alpha} \leftarrow \vec{0}$, $\vec{\theta} \leftarrow \vec{0}$, $\theta_r \leftarrow \infty$          {initialization}
 2: repeat
 3:     $loss \leftarrow 0$
 4:     for $t \leftarrow 1$ to $N$ do
 5:         $\hat{y} \leftarrow \rho(x_t; \vec{\alpha}, \vec{\theta})$          {prediction using Eq. 1}
 6:         if $\hat{y} \neq y_t$ then
 7:             update $loss$ with $|y_t - \hat{y}|$
 8:             $\alpha_t \leftarrow \alpha_t + y_t - \hat{y}$          {weights update}
 9:             for $j \leftarrow \min(\hat{y}, y_t)$ to $\max(\hat{y}, y_t) - 1$ do
10:                 $\theta_j \leftarrow \theta_j - 1$          {thresholds update}
11:             end for
12:         end if
13:     end for
14: until finished($loss$)
15: return ($\vec{\alpha}$, $\vec{\theta}$)
```

It must be assumed that a very large value θ_r corresponds to the largest rank r to ensure the minimum always exists.

A ranking rule partitions $\mathcal{X}$ into $r+1$ equivalence classes corresponding to parallel bands orthogonal w.r.t. $\vec{w}$ and delimited by the thresholds θ_i. Depending on the kernel function adopted κ one may obtain linear or even non linear intervals. Note that the classes need not be equally spaced. Moreover, the objects within each class can further be ordered also by the value of the function $f(x)$.

4.1 The DL-Rank Algorithm

In order to learn the parameters, we adapted the PRank algorithm, a kernelized version of the Perceptron algorithm for Ranking instances expressed as feature vectors [6]. The extended version DL-Rank is defined by rounds (iterations) of the function described in Alg. 1.

The objective of the algorithm is to find a vector of weights $\vec{\alpha}$ which can successfully project all the instances in $\mathcal{X}$ onto the r subintervals defined by the vector of thresholds $\vec{\theta}$, i.e. the subinterval for the j-th rank is $\theta_{j-1} < f(x) < \theta_j$.

The algorithm presented in Alg. 1 may be used in an online fashion, processing the incoming training instances as they are made available. The algorithm continues the iterations starting over until some final condition is met, e.g. a depending on a maximal number of iterations or on the performance on the training instances in terms of average *loss* (see the next section).

At round t the first step is to predict the rank $\hat{y}_t$ (line 4) for a given instance x_t by selecting the smallest rank y such that $f(x) < \theta_y$. If the prediction $\hat{y}_t$ is not the correct rank then the weight vector and the threshold vectors are updated. The rationale is that updating $\vec{\alpha}^t$ and $\vec{\theta}^t$ in this way has the effect of moving the threshold of the desired rank θ_j^{t+1} and the updated predicted rank $f(x_t)$ closer together. This procedure is repeated for all the threshold in the wrong subinterval.

The inner update loop is nested in the outer loop which repeats the optimization of weights and thresholds until a satisfactory rate of training examples is correctly ranked by the resulting function (low or null average loss).

In order to use the ranking algorithm with objects described in a DL knowledge base $\mathcal{K}$, we will consider a training set of $TSet \subseteq \mathsf{Ind}(\mathcal{A})$. Then the related Gram matrix $\vec{K}$ of the kernel values can be obtained by choosing a kernel κ in the family defined in Def. 3.1, $\kappa = k_p^{\mathsf{F}}$, based on a context F of concepts (for example, the leaf concepts of the subsumption hierarchy in $\mathcal{T}$) and a choice of p (e.g. 2).

5. EXPERIMENTS

The DL-Rank system implements the training method and ranking procedure explained in the previous sections, adopting the square kernels of the family (i.e. those with $p = 2$). We aim at directly evaluating the applicability (e.g. in terms of scalability and prediction accuracy) of the proposed method. For this reason, we evaluated the method on a real knowledge base, namely the *Linked User Feedback* (LUF) dataset[3]. LUF is part of an effort to semantically publishing and retrieving user-generated feedbacks (such as ratings, comments and tags); as part of this effort, ratings from the *Linked Movie Data Base*[4] (LinkedMDB) were processed and integrated into the CKAN[5] dataset, which is part of the *Linked Open Data* (LOD) cloud[6].

To evaluate the effectiveness and feasibility of our approach, it was applied to a film ranking prediction task: given a sample of ratings provided by users, the system induces a ranking rule to predict ratings for unranked movies.

In order to leverage the large amount of structured knowledge available through the LOD cloud, we extracted a fragment of the DBpedia [4] knowledge base related to movies ranked in the LUF dataset.

For this task, we followed the procedure described in [13]: starting from resources representing movies, a search was performed in the RDF graph (with recursion depth 1), and up to 1000 superclasses were extracted for each reached object. Such an extraction process resulted in an OWL 2 EL fragment containing 4789 concepts, 59 object properties and 3082 individuals. In the following, we present the experimental setting and discuss the outcomes.

Experimental Setting

We empirically evaluated the effectiveness of the proposed approach by testing its accuracy in predicting movie ratings from the LUF dataset. In this dataset, each movie, represented by its LinkedMDB resource, is related by means of annotations to a set of rankings, that in turn contain the related user and rating value. Therefore, each rating can be considered as a triple $\langle u, m, r\rangle$, where u (resp. m) is an individual representing a user (resp. a movie), and $r \in \mathbf{N}$ is a rating. Ratings range in the set $\{1, \ldots, 5\}$. The histogram in Fig. 1 depicts the distribution of the number of the ratings per user. For each user, represented as an individual in the OWL fragment, a k-fold cross-validation was performed. To emulate the setting of a real-life scenario, where scalability and low time complexity are fundamental, the algorithm was trained in an online fashion: by avoiding it to loop (*finished* predicate in line 14 set to always return `True`), Alg. 1 becomes equivalent to the *kernel perceptron ranking* algorithm (e.g. see [17, p. 262]).

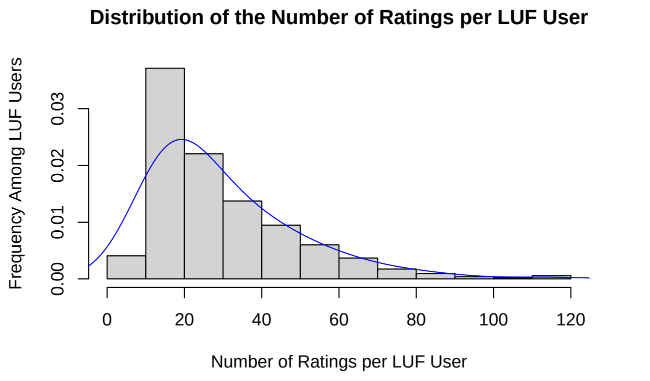

Figure 1: Distribution of the number of ratings per user in the Linked User Feedback dataset.

Beside the proposed kernel, we also evaluated our approach employing another kernel function in the literature, namely the *Full Subtree Kernel* [14]. When showing the experimental results, we indicate this kernel with k_L, whereas we denote as k_F the kernel described in sect. 3. As for the parameters for k_L, the maximum subtree depth is set to 2, and the discount λ is set to 1.0 (as for [14]). We also attempted to transform the obtained kernels using the approach described in [11]; the following two transformations were considered, by making use of the *Polynomial* and *Gaussian* kernels, defined respectively as follows:

- $k_P(x_i, x_j) = (\langle x_i, x_j\rangle + 1)^q, q \in \mathbf{N}$;
- $k_G(x_i, x_j) = exp(-||x_i - x_j||_2^2/2\sigma^2), \sigma \in \mathbf{R}_{++}$,

with $||x_i - x_j||_2^2 = \langle x_i, x_i\rangle - 2\langle x_i, x_j\rangle + \langle x_j, x_j\rangle$. A cross-validation procedure (e.g. a k-fold with $k = 10$) within the training set was used to find the kernel leading to the most accurate results. The q parameter for the polynomial kernel was varied in the range $\{1, \ldots, 9\}$, while the σ parameter for the Gaussian kernel was varied in $\{10^{-4}, 10^{-3}, \ldots, 10^4\}$.

The kernel discussed in Sect. 3 requires a set of concept descriptions F. We evaluated the impact of two choices for such a set: using all the atomic concepts in the ontology (leading to $|\mathsf{F}| = 4789$), which we label with **AC**; and using all the subclasses of three concepts semantically representing films, namely `http://dbpedia.org/ontology/Film`, `http://dbpedia.org/class/yago/Movie106613686` and `http://schema.org/Movie` (which led to $|\mathsf{F}| = 675$), which we label **FSC**. A standard OWL reasoner (Pellet v. 2.3.0) was employed for computing the instance checks.

Results

Tab. 1 reports the experimental results in terms of the *Mean Absolute Error* (MAE) and the *Root Mean Squared Error* (RMSE) indices. From results, it emerges that projecting

[3] `http://thedatahub.org/dataset/linked-user-feedback`
[4] `http://www.linkedmdb.org/`
[5] `http://ckan.org/`
[6] `http://lamboratory.com/2011/12/17/linked-user-feedback/`

Table 1: MAE and RMSE results for each kernel function (Mean $\pm$ Standard Deviation pairs).

	MAE	**RMSE**
k_F (AC)	1.089 ± 0.737	1.243 ± 0.754
k_F (AC)+Gaussian	1.088 ± 0.718	1.25 ± 0.739
k_F (AC)+Polynomial	1.239 ± 0.85	1.407 ± 0.864
	MAE	**RMSE**
k_F (FSC)	1.088 ± 0.738	1.243 ± 0.754
k_F (FSC)+Gaussian	1.076 ± 0.714	1.238 ± 0.737
k_F (FSC)+Polynomial	1.229 ± 0.837	1.395 ± 0.851
	MAE	**RMSE**
k_L	1.726 ± 0.859	1.921 ± 0.862
k_L+Gaussian	1.083 ± 0.731	1.239 ± 0.748
k_L+Polynomial	1.758 ± 0.852	1.953 ± 0.857

the objects (in this case, individuals representing movies in a ranking task) into a higher-dimensional space, led to better results in the discussed online learning task (adopting each user's mean rating as a predictor led to MAE and RMSE values of, respectively, 2.54 ± 0.79 and 2.66 ± 0.75). When used in their basic (linear) form, k_F led to better results than k_L, either with committees composed by all atomic concepts (AC) or all atomic subconcepts of concepts representing films (FSC). A reason for this can be attributed to a number of features considered by the latter kernel (given by the characteristics of the intersection subtree) which are not informative w.r.t. the given ranking task. Also, results suggest that a careful choice of the committee of feature concepts can significantly reduce the complexity of calculating the kernel matrix without degrading the results. We also performed a first empirical evaluation of the adoption of other (batch) ranking methods, not fully reported here for brevity: an one-versus-all ensemble of binary Support Vector Machine (SVM) classifiers and the Soft Ranking algorithm, following the implementations respectively in [17, p. 223, 262]. However, in many occasions it was not possible to find a solution for the aforementioned optimization problem: this factor, jointly with the computational time complexity of traditional large-margin approaches (standard SVM training is $O(m^3)$ in general), led us to adopt an online solution.

6. CONCLUSIONS AND OUTLOOK

The main contribution of this work concerns inductive solutions to the problem of ranking individual resources in DL knowledge bases. Since it is difficult to define a ranking of (retrieved) resources with a general (logical) encoding of preferences, a novel statistical method for learning ranking functions from examples was introduced. The advantages of method based on kernels is that complex (non-linear) ordinal relations can be discovered in the space of individuals, while working on linear models (in the embedding space) with the consequent efficiency. Even more so, the method is suitable for an online utilization, improving the performance of the ranking rule as new ranked instances are available.

Due to the lack of specific testbeds for these specific representation in the related literature, we devised an ad-hoc evaluation method for ranking task: by using the Linked User Feedback (LUF) knowledge base (which is actually part of the LOD cloud), we were able to evaluate the effectiveness of the proposed approach on a real-life dataset of movie ratings. In the near future, it can be foreseen that more knowledge bases containing informations about the ranking of resources w.r.t. some typical usage will be available (e.g. semantically annotated repository of music or movies, etc, with ratings contributed by the users). This will allow the construction of more standard datasets, similarly to related research on ranking based on textual knowledge. The ongoing extensions of the work regard the enhancement of the presented approach by averaging the models obtained in the various update loops according to standard methods, such as *bagging* or *Bayes point* estimation. Besides we are investigating alternative approaches to either content-based or collaborative ranking prediction.

7. REFERENCES

[1] B. Aleman-Meza, C. Halaschek-Wiener, I. Arpinar, C. Ramakrishnan, and A. Sheth. Ranking complex relationships on the semantic web. 9(3):37–44, 2005.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[3] V. Bicer, T. Tran, and A. Gossen. Relational kernel machines for learning from graph-structured rdf data. In *Proc.of the 8th Extended Semantic Web Conference, ESWC 2011b*, volume 6643 of *LNCS*, pages 47–62. Springer, 2011.

[4] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *Web Semant.*, 7(3):154–165, Sept. 2009.

[5] S. Bloehdorn and Y. Sure. Kernel methods for mining instance data in ontologies. In K. Aberer et al., editors, *Proceedings of the 6th International Semantic Web Conference, ISWC2007*, volume 4825 of *LNCS*, pages 58–71. Springer, 2007.

[6] K. Crammer and Y. Singer. Pranking with ranking. In T. Dietterich et al., editors, *Advances in Neural Information Processing Systems 14*, pages 641–647. MIT Press, 2001.

[7] L. Dali, B. Fortuna, and J. Rupnik. Learning to rank for personalized news article retrieval. *Journal of Machine Learning Research - Proceedings Track*, 11:152–159, 2010.

[8] C. d'Amato, N. Fanizzi, and F. Esposito. Query answering and ontology population: An inductive approach. In *Proc. of the 5th European Semantic Web Conference, ESWC2008*, volume 5021 of *LNCS*, pages 288–302. Springer, 2008.

[9] T. Di Noia, E. Di Sciascio, and F. Donini. Semantic matchmaking as non-monotonic reasoning: A description logic approach. *Journal of Artificial Intelligence Research*, 29:269–307, 2007.

[10] N. Fanizzi, C. d'Amato, and F. Esposito. Induction of robust classifiers for web ontologies through kernel machines. *J. Web Sem.*, 11:1–13, 2012.

[11] M. Gönen and E. Alpaydin. Multiple kernel learning algorithms. *J. Mach. Learn. Res.*, 12:2211–2268, July 2011.

[12] E. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In T. Fawcett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning, ICML2003*, pages 250–257. AAAI Press, 2003.

[13] S. Hellmann, J. Lehmann, and S. Auer. Learning of owl class descriptions on very large knowledge bases. *International Journal On Semantic Web and Information Systems*, 2009.

[14] U. Lösch, S. Bloehdorn, and A. Rettinger. Graph kernels for rdf data. In E. Simperl, P. Cimiano, A. Polleres, Ó. Corcho, and V. Presutti, editors, *ESWC*, volume 7295 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2012.

[15] T. Lukasiewicz and J. Schellhase. Variable-strength conditional preferences for ranking objects in ontologies. *Web Semantics*, 5(3):180–194, 2007.

[16] J. Pan, G. Stamou, G. Stoilos, S. Taylor, and E. Thomas. Scalable querying services over fuzzy ontologies. In *Proc. of the Int. World Wide Web Conference, WWW2008*, pages 575–584. ACM, 2008.

[17] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[18] U. Straccia. Towards top-k query answering in description logics: The case of DL-Lite. In *Proc. of the 10th European Conf. on Logics in Artificial Intelligence, JELIA2006*, volume 4160 of *LNCS*, pages 439–451. Springer, 2006.