

Towards Learning to Rank in Description Logics

Nicola Fanizzi, Claudia d’Amato, Floriana Esposito¹

Abstract. In the context of knowledge bases expressed in Description Logics, a method for learning functions that can predict the ranking of resources encoding some preference criteria implicitly encoded through examples of rated individuals. The method relies on a kernelized version of the PERCEPTRON RANKING algorithm which is suitable for batch but also online problem settings.

1 INTRODUCTION

When querying a knowledge base (henceforth KB), likely the answers may be expected to be returned ranked w.r.t. some criteria. This is a hard task, especially when it is difficult to define a general and precise notion of the relevance w.r.t. the criteria of choice (e.g. multiple relevance orders, preferences, etc.). Still it may be possible to express such information imprecisely by means of examples rather than explicitly. A related relevance-feedback mechanism is the task known as *collaborative filtering* which aims at detecting the relevance of information items for a new user based on rankings previously acquired from others.

All such similar problems can be cast in the framework of *learning from examples*. Given previously rated items w.r.t. some criterion that can be hardly formally encoded, the aim is to learn a function which could be then used to assign a meaningful rating to new incoming instances of uncertain. We tackle this problem in the context of mining ontologies expressed in Description Logics (DLs).

Solutions to related problems have been given based on the structure of the descriptions (e.g. see [1]) failing to fully exploit the representation expressiveness. Related logical methods, although elegant and well-founded, may fall short in terms of scalability and language-dependence [3]. Other frameworks tackle the problem by explicitly representing the preferences with conditional statements for ranking objects [6]. We recur to statistical learning methods which have been employed for inductive classification tasks [4]. Indeed learning to rank can be cast as an inductive problem, to be solved by induce a ranking function from a training set of ranked individuals.

2 RANKING PROBLEMS

One may define *learning to rank* 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 endowed with a total order relation $\leq_Y$, such that $|Y| = r \in \mathbb{N}$ (discrete case) or $Y = \mathbb{R}^+$ (continuous case). The goal is to derive a function to predict the rank for new unlabeled instances. Formally:

given $S = \{(x_1, y_1), \dots, (x_N, y_N)\} \subseteq \mathcal{X} \times Y$,

where $y_i \in Y$ is the rank of $x_i \in \mathcal{X}$

find a ranking function $\rho : \mathcal{X} \mapsto Y$

that induces a partial (resp. total) order $\preceq_\rho$ over $\mathcal{X}$.

The simplest way to define the set of ranks in the continuous case is $Y = \mathbb{R}^+$ or, in the discrete case, $Y = \{1, \dots, r\}$. In this setting, one may say that x_i is *preferred* over x_j , denoted $x_i \preceq_\rho x_j$, iff $\rho(x_i) \leq_Y \rho(x_j)$.

Learning a ranking function may be easily cast a special *classification* problem considering ranks in a discrete scale or even as a natural *regression* problem. However, in the former case the information regarding the order among the labels would be overlooked; conversely, mapping the discrete ranks to real values, for resorting to a regression setting, would require specific metrics that may make the process very sensitive to the representation of the ranks rather than to their ordering [5]. Therefore, it is preferable to treat ranking as a problem in its own right and design specific algorithms that take advantage of the specific nature of that problem [7].

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.

3 RANK LEARNING WITH DL KERNELS

A learning task generally requires finding a function h (*hypothesis*) which can be adopted by a decision procedure to predict, given an input instance, the correct value for the target complex function to be approximated. Ideally, h should be also efficiently evaluable (e.g. linear) and determine a close approximation. To such purpose, kernel methods [7] are particularly well suited because the learning algorithm and the choice of h are almost completely independent. Moreover, the model boils down to discovering linear relations. Kernel function make it possible to represent such models efficiently in high-dimensional spaces. The related learning algorithms are known to be general-purpose, robust and also efficient: they require a computational effort that is polynomial in the size of the data items, even when the dimension of the embedding space grows exponentially. Examples of inputs and expected outputs are to be provided to the algorithm to produce a definition for the target function in the form of a linear decision function depending on a tuple of weights.

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.

We resort to a set of kernels for individuals in DL KBs [4], that exploit a notion of similarity in the context of a set of concepts:

Definition 1 (DL kernel functions) Given a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and a set of concept descriptions

¹ University of Bari, Italy, email: {fanizzi,claudia.damato,esposito}@di.uniba.it

$F = \{F_1, F_2, \dots, F_m\}$ defined in $\mathcal{K}$, the kernel functions based on F , are mappings $k_p^F: \text{Ind}(\mathcal{A}) \times \text{Ind}(\mathcal{A}) \mapsto [0, 1]$ defined ($p \in \mathbb{R}_+$) as follows $\forall a, b \in \text{Ind}(\mathcal{A})$ $k_p^F(a, b) = [\sum_{i=1}^m (\kappa_i(a, b))^p]^{1/p}$ where $\forall i \in \{1, \dots, m\}$: κ_i is defined:

$$\kappa_i(a, b) = \begin{cases} 1 & \mathcal{K} \models F_i(a) \wedge \mathcal{K} \models F_i(b) \vee \mathcal{K} \models \neg F_i(a) \wedge \mathcal{K} \models \neg F_i(b) \\ 0 & \mathcal{K} \models \neg F_i(a) \wedge \mathcal{K} \models F_i(b) \vee \mathcal{K} \models F_i(a) \wedge \mathcal{K} \models \neg F_i(b) \\ u_i & \text{otherwise} \end{cases}$$

The rationale behind these kernels is that similarity between individuals is determined w.r.t. each concept in a given context of features. Two individuals are maximally similar w.r.t. a given concept F_i if they exhibit the semantic same behavior; conversely, minimal similarity is assigned when they belong to disjoint concepts. Because of the OWA, concept-membership cannot be always determined, hence an intermediate value $u_i \in (0, 1)$ is assigned to reflect such uncertainty. This value should estimate the probability that two instances may belong (not belong) to the extension of F_i based on the sizes of the retrieval sets of F_i and its complement w.r.t. the current ABox (a .5 value is considered in case of total uncertainty).

Kernel-based Ranking. Preliminarily, we will assume an implicit kernel-defined *feature space* with the corresponding mapping ϕ so that $\phi(x_i)$ is in $\mathbb{R}^n$ for some n , $1 \leq n \leq \infty$ (see [7] for details). A *linear ranking rule* function embeds the input data into the real axis by means of a linear function in the feature space $f(x) = (\mathbf{w} \cdot \phi(x))$. The returned real-value can be then converted to a rank by means of an r -dimensional vector of thresholds θ 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 2 (ranking rule) Given a kernel $\kappa: \mathcal{X} \mapsto \mathbb{R}$, the ranking of $x \in \mathcal{X}$ is given² by $\rho(x; \mathbf{w}, \theta) = \min\{y \in Y \mid f(x) < \theta_y\}$ that is, in a dual representation, where $\mathbf{w} = \sum_{i=1}^N \alpha_i \phi(x_i)$:

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

Such a ranking rule partitions $\mathcal{X}$ into $r+1$ equivalence classes corresponding to parallel bands orthogonal w.r.t. $\mathbf{w}$ and delimited by the thresholds θ_i . Depending on κ , one may obtain linear or even non-linear intervals. Note also 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 $f(x)$.

In order to learn the parameters of a ranking function, we devised DL-KPRANK, a kernelized version of the PERCEPTRON RANKING algorithm PRANK [2]. The core of the DL-KPRANK algorithm is defined in terms of iterations of the KPRANKUPDATE function shown in Fig. 1. The objective of the algorithm is to find a PERCEPTRON weight vector α which successfully projects the elements of $\mathcal{X}$ into the r subintervals defined by the thresholds θ , i.e. the subinterval for the rank of j is $\theta_{j-1} < f(x) < \theta_j$.

At round t the first step predicts the rank $\hat{y}_t$ (line 2) for x_t by selecting the smallest rank y such that $f(x) < \theta_y$. If the prediction $\hat{y}_t$ is not correct then a label of $\ell_j^t = +1$ is allocated to those subintervals above the target rank y_t and $\ell_j^t = -1$ to those below (lines 3, 4 and 5). For each subinterval, if the ranking rule based on α^t and θ^t misclassifies the label ℓ_j^t then the label is subtracted from the threshold θ_j^t , and the PERCEPTRON weights are updated. The rationale is that updating α^t and θ^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 subintervals $j = 1, \dots, r-1$ at round t .

The call to the update routine is inserted in an outer loop which repeats the weight and threshold optimization until a satisfactory rate of

function KPRANKUPDATE($x_t, y_t, \alpha^t, \theta^t$): ($\alpha^{t+1}, \theta^{t+1}$)
 (x_t, y_t) : ranked training example at round t
 α^t, θ^t : weights at rounds $t, t+1$
 θ^t, θ^{t+1} : thresholds at rounds $t, t+1$

```

1:  $\mathbf{u}^t \leftarrow \mathbf{0}, \ell^t \leftarrow \mathbf{0}$  {initialization}
2:  $\hat{y} \leftarrow \rho(x_t; \alpha^t, \theta^t)$  {prediction}
3: if  $\hat{y} \neq y_t$  then
4:   for  $j = 1$  to  $r-1$  do
5:     if  $j \geq y_t$  then  $\ell_j^t \leftarrow -1$  else  $\ell_j^t \leftarrow +1$  endif
6:   end for
7:   for  $j = 1$  to  $r-1$  do
8:     if  $(\sum_{i=1}^N \alpha_i^t \kappa(x_t, x_i) - \theta_j^t) \ell_j^t \leq 0$  then  $u_j^t \leftarrow \ell_j^t$  endif
9:   end for
10:   $\alpha^{t+1} \leftarrow \alpha^t + y_t \sum_{j=1}^{r-1} u_j^t$  {weights update}
11:   $\theta^{t+1} \leftarrow \theta^t - \mathbf{u}^t$  {thresholds update}
12: else
13:   $\alpha^{t+1} \leftarrow \alpha^t$ 
14:   $\theta^{t+1} \leftarrow \theta^t$ 
15: end if
16: return ( $\alpha^{t+1}, \theta^{t+1}$ )

```

Figure 1. The core updating function of the DL-KPRANK algorithm.

training examples is correctly ranked by the resulting function (low or null average loss). In order to use the algorithm with objects described in a KB $\mathcal{K}$, we consider a training set of $TSet \subseteq \text{Ind}(\mathcal{A})$. Then the kernel matrix $\mathbf{K}$ is obtained using a kernel κ from the family in Def. 1, based on a context F of concepts (e.g., the leaf concepts of the subsumption hierarchy in $\mathcal{T}$) and a choice of p (e.g. 2).

4 CONCLUSIONS & OUTLOOK

We have presented a solution based on statistical learning to the problem of ranking individuals in DL KBs. 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. Even more so, the method is suitable for an online utilization, improving its performance as new ranked instances are available.

The extensions of the work will concern the investigation of alternative algorithms derived from support vector regression and the enhancement of the presented one by averaging the models obtained in the various update loops according to standard methods, such as *bagging* or *Bayes point* estimation.

REFERENCES

- [1] B. Aleman-Meza, C. Halaschek-Wiener, I. Arpinar, C. Ramakrishnan, and A.P. Sheth, 'Ranking complex relationships on the semantic web', *IEEE Internet Computing*, **9**(3), 37–44, (2005).
- [2] K. Crammer and Y. Singer, 'Ranking with ranking', in *Advances in Neural Information Processing Systems 14*, eds., T. Dietterich et al., pp. 641–647. MIT Press, (2001).
- [3] T. Di Noia, E. Di Sciascio, and F.M. Donini, 'Semantic matchmaking as non-monotonic reasoning: A description logic approach', *J. Artif. Intell. Res.*, **29**, 269–307, (2007).
- [4] N. Fanizzi, C. d'Amato, and F. Esposito, 'Statistical learning for inductive query answering on OWL ontologies', in *Proc. of ISWC2008*, eds., A. Sheth et al., volume 5318 of *LNCS*, pp. 195–212. Springer, (2008).
- [5] E.F. Harrington, 'Online ranking/collaborative filtering using the perceptron algorithm', in *Proc. of ICML2003*, eds., T. Fawcett and N. Mishra, pp. 250–257. AAAI Press, (2003).
- [6] T. Lukasiewicz and J. Schellhase, 'Variable-strength conditional preferences for ranking objects in ontologies', *Web Semant.*, **5**(3), 180–194, (2007).
- [7] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*, Cambridge University Press, 2004.

² A large $\theta_r \approx +\infty$ is chosen to ensure the existence of the minimum.