Mach Learn (2018) 107:903–941
https://doi.org/10.1007/s10994-017-5694-9

Dyad ranking using Plackett–Luce models based on joint feature representations

Dirk Schäfer[1] · **Eyke Hüllermeier**[1]

Received: 25 May 2017 / Accepted: 21 December 2017 / Published online: 10 January 2018

Abstract Label ranking is a specific type of preference learning problem, namely the problem of learning a model that maps instances to rankings over a finite set of predefined alternatives. Like in conventional classification, these alternatives are identified by their name or *label* while not being characterized in terms of any properties or features that could be potentially useful for learning. In this paper, we consider a generalization of the label ranking problem that we call *dyad ranking*. In dyad ranking, not only the instances but also the alternatives are represented in terms of attributes. For learning in the setting of dyad ranking, we propose an extension of an existing label ranking method based on the Plackett–Luce model, a statistical model for rank data. This model is combined with a suitable feature representation of dyads. Concretely, we propose a method based on a bilinear extension, where the representation is given in terms of a Kronecker product, as well as a method based on neural networks, which allows for learning a (highly nonlinear) joint feature representation. The usefulness of the additional information provided by the feature description of alternatives is shown in several experimental studies. Finally, we propose a method for the visualization of dyad rankings, which is based on the technique of multidimensional unfolding.

Keywords Preference learning · Label ranking · Dyad ranking · Plackett–Luce model · Neural networks · Multidimensional unfolding

Editor: Henrik Boström.

✉ Eyke Hüllermeier
eyke@upb.de

Dirk Schäfer
dirk.schaefer@jivas.de

[1] Department of Computer Science, Paderborn University, Paderborn, Germany

1 Introduction

Preference learning is an emerging subfield of machine learning, which deals with the induction of preference models from observed or revealed preference information (Fürnkranz and Hüllermeier 2010). Such models are typically used for prediction purposes, for example to predict context-dependent preferences of individuals on various choice alternatives. Depending on the representation of preferences, individuals, alternatives, and contexts, a large variety of preference models are conceivable, and many such models have already been studied in the literature.

A specific type of preference learning problem is the problem of *label ranking*, namely the problem of learning a model that maps instances to rankings (total orders) over a finite set of predefined alternatives (Vembu and Gärtner 2010). An instance, which defines the context of the preference relation, is typically characterized in terms of a set of attributes or features; for example, an instance could be a person described by properties such as sex, age, income, etc. As opposed to this, the alternatives to be ranked, e.g., the political parties of a country, are only identified by their name (label), while not being characterized in terms of any properties or features.

In this paper, we introduce *dyad ranking* as a generalization of the label ranking problem. In dyad ranking, not only the instances but also the alternatives are represented in terms of attributes. For learning in the setting of dyad ranking, we propose an extension of an existing label ranking method based on the Plackett–Luce (PL) model, a statistical model for rank data (Luce 1959; Plackett 1975). The extension essentially consists of expressing the parameters of this model as functions of a suitably defined *joint* feature representation of dyads.

To this end, we propose two approaches. The first one is based on a bilinear extension of an existing linear version of the PL model, where the representation is given in terms of a Kronecker product (Schäfer and Hüllermeier 2015). The second one is much more flexible and makes use of a neural network that allows for learning a highly nonlinear joint feature representation. The usefulness of the additional information provided by the feature description of alternatives is shown in several experimental studies.

Another contribution of the paper is a method for the visualization of dyad rankings, which is based on the technique of multidimensional unfolding. While this technique has been used in statistics for quite a while, it has not received much attention in machine learning so far (Murata et al. 2017).

The paper is organized as follows. In the next section, we introduce the problem of dyad ranking. Following a discussion of related problem settings in Sect. 3, we propose the joint-feature Plackett–Luce model in Sect. 4. In Sects. 5 and 6, we introduce two instantiations of this model, the first based on the Kronecker product for representing dyads, and the second making use of neural networks. In Sect. 7, we propose a method for the visualization of dyad rankings, which is based on the technique of multidimensional unfolding. All methods are evaluated experimentally in Sect. 8.

2 Dyad ranking

As will be explained in more detail later on (cf. Sect. 3), the learning problem addressed in this paper has connections to several existing problems in the realm of preference learning. In particular, it can be seen as a combination of *dyadic prediction* (Menon and Elkan 2010a, b, c) and *label ranking* (Vembu and Gärtner 2010), hence the term "dyad ranking". Since our

method for tackling this problem is an extension of a label ranking method, we will introduce dyad ranking here as an extension of label ranking.

2.1 Label ranking

Let $\mathcal{Y} = \{y_1, \ldots, y_K\}$ be a finite set of (choice) alternatives; adhering to the terminology commonly used in supervised machine learning, and accounting for the fact that label ranking can be seen as an extension of multi-class classification, the y_i are also called *class labels* or simply *labels*. We consider total order relations $\succ$ on $\mathcal{Y}$, that is, complete, transitive, and antisymmetric relations, where $y_i \succ y_j$ indicates that y_i precedes y_j in the order. Since a ranking can be seen as a special type of preference relation, we shall also say that $y_i \succ y_j$ indicates a preference for y_i over y_j. We interpret this order relation in a wide sense, so that $a \succ b$ can mean that the alternative a is more liked by a person than alternative b, but also for example that an algorithm a outperforms algorithm b.

Formally, a total order $\succ$ can be identified with a permutation π of the set $[K] = \{1, \ldots, K\}$, such that $\pi(i)$ is the index of the label on position i in the permutation. We denote the class of permutations of $[K]$ (the symmetric group of order K) by $\mathbb{S}_K$. By abuse of terminology, though justified in light of the above one-to-one correspondence, we refer to elements $\pi \in \mathbb{S}_K$ as both permutations and rankings.

In the setting of label ranking, preferences on $\mathcal{Y}$ are "contextualized" by instances $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ is an underlying instance space. Thus, each instance $\boldsymbol{x}$ is associated with a ranking $\succ_{\boldsymbol{x}}$ of the label set $\mathcal{Y}$ or, equivalently, a permutation $\pi_{\boldsymbol{x}} \in \mathbb{S}_K$. More specifically, since label rankings do not necessarily depend on instances in a deterministic way, each instance $\boldsymbol{x}$ is associated with a probability distribution $\mathbf{P}(\cdot \mid \boldsymbol{x})$ on $\mathbb{S}_K$. Thus, for each $\pi \in \mathbb{S}_K$, $\mathbf{P}(\pi \mid \boldsymbol{x})$ denotes the probability to observe the ranking π in the context specified by $\boldsymbol{x}$.

As an illustration, suppose $\mathcal{X}$ is the set of people characterized by attributes such as sex, age, profession, and marital status, and labels are music genres: $\mathcal{Y} = \{\texttt{Rock}, \texttt{Pop}, \texttt{Classic}, \texttt{Jazz}\}$. Then, for $\boldsymbol{x} = (m, 30, \text{teacher}, \text{married})$ and $\pi = (2, 1, 3, 4)$, $\mathbf{P}(\pi \mid \boldsymbol{x})$ denotes the probability that a 30 years old married man, who is a teacher, prefers Pop music to Rock to Classic to Jazz.

The goal in label ranking is to learn a "label ranker", that is, a model

$$h : \mathcal{X} \longrightarrow \mathbb{S}_K$$

that predicts a ranking π for each instance $\boldsymbol{x}$ given as an input. More specifically, seeking a model with optimal prediction performance, the goal is to find a risk (expected loss) minimizer

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \int_{\mathcal{X} \times \mathbb{S}_K} \mathcal{L}(h(\boldsymbol{x}), \pi)\, d\,\mathbf{P},$$

where $\mathcal{H}$ is the underlying hypothesis space, $\mathbf{P}$ is the joint measure $\mathbf{P}(\boldsymbol{x}, \pi) = \mathbf{P}(\boldsymbol{x})\mathbf{P}(\pi \mid \boldsymbol{x})$ on $\mathcal{X} \times \mathbb{S}_K$, and $\mathcal{L}$ is a loss function on $\mathbb{S}_K$.

As training data $\mathcal{D}$, a label ranker uses a set of instances $\boldsymbol{x}_n$ ($n \in [N]$), together with information about the associated rankings π_n. Ideally, complete rankings are given as training information, i.e., a single observation is a tuple of the form $(\boldsymbol{x}_n, \pi_n) \in \mathcal{X} \times \mathbb{S}_K$. From a practical point of view, however, it is important to allow for incomplete information in the form of a ranking of some but not all of the labels in $\mathcal{Y}$:

$$y_{\pi(1)} \succ_{\boldsymbol{x}} y_{\pi(2)} \succ_{\boldsymbol{x}} \cdots \succ_{\boldsymbol{x}} y_{\pi(J)}, \tag{1}$$

where $J \leq K$ and $\{\pi(1), \ldots, \pi(J)\} \subset [K]$. For example, for an instance $\boldsymbol{x}$, it might be known that $y_2 \succ_{\boldsymbol{x}} y_1 \succ_{\boldsymbol{x}} y_5$, while no preference information is given about the labels y_3 or y_4.

2.2 Dyad ranking as an extension of label ranking

In the setting of label ranking as introduced above, instances are supposed to be characterized in terms of properties—typically, an instance is represented as an r-dimensional feature vector $\boldsymbol{x} = (x_1, \ldots, x_r)$. As opposed to this, the alternatives to be ranked, the labels y_i, are only identified by their name, just like categories in classification.

Needless to say, a learner may benefit from knowledge about properties of the alternatives, too. In fact, if the preferences of an instance are somehow connected to such properties, then alternatives with similar properties should also be ranked similarly. In particular, by sharing information via features, it would in principle be possible to rank alternatives that have never be seen in the training process so far, i.e., to do some kind of "zero-shot learning" (Larochelle et al. 2008).

Returning to our above example of ranking music genres, suppose we know (or at least are quite sure) that `Rock` $\succ_{\boldsymbol{x}}$ `Classic` $\succ_{\boldsymbol{x}}$ `Jazz` for a person $\boldsymbol{x}$. We would then expect that `Pop` is ranked more likely close to the top than close to the bottom, simply because Pop music is more similar to Rock than to Classic or Jazz. In contrast to a label ranker, for which the music genres are just uninformative names, we are able to make a prediction of that kind thanks to our knowledge about the different types of music.

Given that useful properties of alternatives are indeed often available in practice, we introduce dyad ranking as an extension of label ranking, in which alternatives are elements of a feature space:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_c) \in \mathcal{Y} = \mathcal{Y}_1 \times \mathcal{Y}_2 \times \cdots \times \mathcal{Y}_c, \tag{2}$$

where $\mathcal{Y}_i \subseteq \mathbb{R}$, $1 \leq i \leq c$. Then, a *dyad* is a pair

$$\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{Z} = \mathcal{X} \times \mathcal{Y} \tag{3}$$

consisting of an instance $\boldsymbol{x}$ and an alternative $\boldsymbol{y}$. We assume training information to be given in the form of rankings

$$\rho_i : \boldsymbol{z}^{(1)} \succ \boldsymbol{z}^{(2)} \succ \cdots \succ \boldsymbol{z}^{(M_i)} \tag{4}$$

of a finite number of dyads, where M_i is the length of the ranking. Typically, though not necessarily, all dyads in (4) share the same context $\boldsymbol{x}$, i.e., they are all of the form $\boldsymbol{z}^{(j)} = (\boldsymbol{x}, \boldsymbol{y}^{(j)})$; in this case, (4) can also be written as

$$\rho_i : \boldsymbol{y}^{(1)} \succ_{\boldsymbol{x}} \boldsymbol{y}^{(2)} \succ_{\boldsymbol{x}} \ldots \succ_{\boldsymbol{x}} \boldsymbol{y}^{(M_i)}. \tag{5}$$

Likewise, a prediction problem will typically consist of ranking a subset

$$\left\{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \ldots, \boldsymbol{y}^{(M)}\right\} \subseteq \mathcal{Y}$$

in a given context $\boldsymbol{x}$. Being provided with a *dyad ranker*, i.e., a model that produces a ranking of dyads as an output, this can be accomplished by applying that ranker to the set of dyads

$$\left(\boldsymbol{x}, \boldsymbol{y}^{(1)}\right), \left(\boldsymbol{x}, \boldsymbol{y}^{(2)}\right), \ldots, \left(\boldsymbol{x}, \boldsymbol{y}^{(M)}\right)$$

and then projecting the result to the alternatives, i.e., transforming a ranking of the form (4) into one of the form (5). This setting, which generalizes label ranking in the sense that

additional information in the form of feature vectors is provided for the labels, is the main subject of this paper and will subsequently be referred to as *contextual dyad ranking*.

Note that (5) covers the important case of pairwise comparisons as a special case ($M_i = 2$). Pairwise comparisons are especially appealing in subjective preference judgments (David 1969), and hence can be motivated from this point of view. Moreover, within machine learning, the case of pairwise preferences has been studied quite extensively, because it allows for reducing the problem of ranking to simpler learning problems, such as binary classification (Dekel et al. 2004).

3 Related settings

As already mentioned earlier, the problem of dyad ranking is not only connected to label ranking, but also to several other types of ranking and preference learning problems that have been discussed in the literature.

The term "dyad ranking" derives from the framework of *dyadic prediction* as introduced by Menon and Elkan (2010b). This framework can be seen as a generalization of the setting of collaborative filtering (CF), in which *row-objects* (e.g., clients) are distinguished from *column-objects* (e.g., products). Moreover, with each combination of such objects, called a dyad by Menon and Elkan, a value (e.g., a rating) is associated. While in CF, row-objects and column-objects are only represented by their name (just like the alternatives in label ranking), they are allowed to have a feature representation (called side-information) in dyadic prediction. Menon and Elkan are trying to exploit this information to improve performance in matrix completion, i.e., predicting the values for those object combinations that have not been observed so far, in very much the same way as we are trying to make use of feature information in the context of label ranking.

CF and ranking are combined in collaborative ranking. Here, the aim is to provide personalized recommendations for users in the form of rankings on items (Weimer et al. 2007). In contrast to CF, where rankings can be obtained indirectly by sorting items according to predicted scores (which potentially leads to many ties), collaborative ranking tackles the ranking problem more directly.

Methods for *learning-to-rank* or *object ranking* (Cohen et al. 1999; Kamishima et al. 2011) have received a lot of attention in the recent years, especially in the field of information retrieval (Liu 2011). In general, the goal is to learn a *ranking function* that accepts a subset $O \subset \mathcal{O}$ of objects as input, where $\mathcal{O}$ is a reference set of objects (e.g., the set of all books). As output, the function produces a ranking (total order) $\succ$ of the objects O. The ranking function is commonly implemented by means of a scoring function $U : \mathcal{O} \longrightarrow \mathbb{R}$, i.e., objects are first scored and then ranked according to their scores (Hüllermeier and Vanderlooy 2009). In order to induce a function of that kind, the learning algorithm is provided with training information, which typically comes in the form of exemplary pairwise preferences between objects. As opposed to label ranking, the alternatives to be ranked are described in terms of properties (feature vectors), while preferences are not contextualized. In principle, methods for object ranking could be applied in the context of dyad ranking, too, namely by equating the object space $\mathcal{O}$ with the "dyad space" $\mathcal{Z}$ in (3); in fact, dyads can be seen as a specific type of object, i.e., as objects with a specific structure. Especially close in terms of the underlying methodology is the so-called *listwise approach* in learning-to-rank (Cao et al. 2007). We elaborate on the relation to approaches of that kind in more detail in Sect. 6.

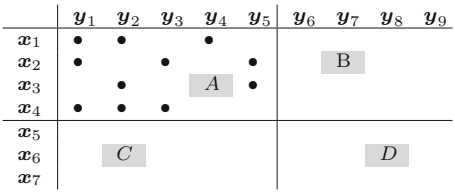

Fig. 1 Problems of type A (upper left), B (upper right), C (lower left), D (lower right). Dyads marked by a bullet are part of the training data. Predictions are sought for dyads marked by a shaded field

Conditional ranking (CR) approaches preference modeling from a graph-theoretic perspective (Pahikkala et al. 2010, 2013). Relational data is considered as a multi-graph, where nodes correspond to objects that are represented by attributes. Edges between nodes are weighted by values $y \in [0, 1]$, each one expressing a binary relation (strength of preference) between two nodes (Pahikkala et al. 2010). A pair of nodes can be connected with multiple edges, thereby allowing one to express repeated measurements. The aim of conditional ranking is the prediction of a node ranking relative to (or "conditioned on") a reference node. The CR methods proposed by Pahikkala et al. (2010, 2013) are based on (efficient) variations of a kernel-supported regularized least-squares approach (RankRLS). To utilize them, it is necessary to extract contextual pairwise training data from the graph. These are identified by three connected nodes in the graph as follows:

$$v' \succeq_v v'' \Leftrightarrow Q(v, v') \geq Q(v, v''), \tag{6}$$

where $Q(a, b)$ is an edge weight between the nodes a and b. The differences $Q(v, v') - Q(v, v'')$ are used as target variables for learning a regression function. More specifically, RankRLS starts from the disagreement error (the number of violations of (6), i.e., with $<$ instead of $\geq$ on the right-hand side) and uses the squared error as a surrogate. Thus, the loss on a preference (6) would be given by $\left(1 - \left[Q(v, v') - Q(v, v'')\right]\right)^2$. Despite being easy to optimize, note that this loss is not monotone decreasing in the score difference $Q(v, v') - Q(v, v'')$, as it actually should.

In most applications of CR so far, all nodes of the graph stem from the same domain, i.e., only a single type of object is considered (Pahikkala et al. 2010, 2013). The framework as such is more general, however, and the graph can also be bipartite. Then, v in (6) has a different type than v' and v'', which means that a tuple (v, v') can be seen as a dyad. From this point of view, CR is indeed very close to dyad ranking, also because the kernel function used in CR is based on joint feature representations (Pahikkala et al. 2013).

In comparison to CR, let us highlight the *probabilistic* and *listwise* nature as important features of our approach. Probabilistic predictions allow for representing the uncertainty in a prediction, which often turns out to be useful (as will be seen, for example, in Sects. 5.3.3 and 8) and provides the basis for extended problems settings, such as ranking with abstention (Cheng et al. 2010b, 2012). Besides, the underlying probabilistic model is amenable to principled learning techniques such as maximum likelihood. The second property allows for making use of complete rankings as training information, with pairwise preferences (6) only being a special case. It avoids the need to break a ranking into a set of pairwise comparisons, which necessarily comes with a loss of information that may lead to biased estimations, especially for the PL model (Soufiani et al. 2014).

The authors of CR distinguish four different prediction problems for dyadic data, which are illustrated in Fig. 1 (Pahikkala et al. 2014). Problem A refers to the situation in which a prediction is sought for a dyad $(\boldsymbol{x}, \boldsymbol{y})$, such that both $\boldsymbol{x}$ and $\boldsymbol{y}$ have already been encountered in the training data, though not necessarily in combination ($(\boldsymbol{x}_3, \boldsymbol{y}_4)$ in the example). In contrast to this, problem D asks for a prediction on a dyad $(\boldsymbol{x}, \boldsymbol{y})$ such that neither $\boldsymbol{x}$ nor

$\boldsymbol{y}$ is contained in the training data (*zero-shot learning*, $(\boldsymbol{x}_6, \boldsymbol{y}_8)$ in the example). Problems B and C are in-between: either $\boldsymbol{x}$ or $\boldsymbol{y}$ has already been encountered, but not both ($(\boldsymbol{x}_2, \boldsymbol{y}_7)$, $(\boldsymbol{x}_6, \boldsymbol{y}_2)$ in the example). A prediction in dyad ranking, which involves several dyads, can be a mixture of these situations. Its flexibility is clearly a strength of the framework: It is applicable to problems of type A and C, and provided predictive features are available on Y, also to settings B and D.

Finally, we note that dyad ranking deals with predictions (rankings of dyads) having a complex structure. Therefore, like for ranking and preference learning in general, there is also a natural connection to the field of structured output prediction (Bakir et al. 2007).

4 Joint-feature Plackett–Luce models

4.1 The basic model

The Plackett–Luce (PL) model is a parameterized probability distribution on the set of all rankings over a set of alternatives $y_1, \ldots, y_K$. It is specified by a parameter vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_K) \in \mathbb{R}_+^K$, in which v_i accounts for the "skill" (latent utility) of the option y_i. The probability assigned by the PL model to a ranking represented by a permutation π is given by

$$\mathbf{P}(\pi \mid \boldsymbol{v}) = \prod_{i=1}^{K-1} \frac{v_{\pi(i)}}{v_{\pi(i)} + v_{\pi(i+1)} + \cdots + v_{\pi(K)}}. \tag{7}$$

This model is a generalization of the well-known Bradley–Terry model (Marden 1995), a model for the pairwise comparison of alternatives, which specifies the probability that "a wins against b" in terms of

$$\mathbf{P}(a \succ b) = \frac{v_a}{v_a + v_b}. \tag{8}$$

Obviously, the larger v_a in comparison to v_b, the higher the probability that a is chosen. Likewise, the larger the parameter v_i in (7) in comparison to the parameters v_j, $j \neq i$, the higher the probability that y_i appears on a top rank.

An intuitively appealing explanation of the PL model can be given in terms of a vase model: If v_i corresponds to the relative frequency of the ith label in a vase filled with labeled balls, then $\mathbf{P}(\pi \mid \boldsymbol{v})$ is the probability to produce the ranking π by randomly drawing balls from the vase in a sequential way and putting the label drawn in the kth trial on position k (unless the label was already chosen before, in which case the trial is annulled). This explanation corresponds to the interpretation of the model as being a multistage model.

A nice feature of the Plackett–Luce model is that marginals (i.e., probabilities of rankings of a subset of the alternatives) can be computed very easily for this model: The probability of an incomplete ranking

$$y_{\pi(1)} \succ y_{\pi(2)} \succ \cdots \succ y_{\pi(J)}$$

is given by

$$\mathbf{P}(\pi \mid \boldsymbol{v}) = \prod_{i=1}^{J} \frac{v_{\pi(i)}}{v_{\pi(i)} + v_{\pi(i+1)} + \cdots + v_{\pi(J)}}, \tag{9}$$

i.e., by an expression of exactly the same form as (7), except that the number of factors is J instead of K. Note that we recover (8) as a special case of (9) for $J = 2$.

As an aside, we note that PL is arguably the most well-known model in the field of statistics for rank data, next to the Mallows model (Mallows 1957). The latter is a distance-based model that belongs to the family of exponential distributions. It is specified by a location parameter $\pi_0 \in \mathbb{S}_K$ and a spread parameter $\theta \geq 0$:

$$\mathbf{P}(\pi \mid \theta, \pi_0) = \frac{1}{\phi(\theta)} \exp\big(-\theta \cdot D(\pi, \pi_0)\big), \tag{10}$$

where D is a distance function on $\mathbb{S}_K$ (typically the Kendall distance) and $\phi(\theta)$ a normalization constant. Both models, Mallows and PL, have been used in the setting of label ranking (Cheng et al. 2009). In general, Mallows is a bit more difficult to handle, for example as it does not have simple closed-form expressions for the probabilities of incomplete rankings (Cheng et al. 2010a).

4.2 Plackett–Luce model with features

The use of PL for label ranking, as proposed by Cheng et al. (2010a), is to contextualize the skill parameters v_k of the labels y_k by modeling them as functions of the context $\boldsymbol{x}$. More precisely, to guarantee the non-negativity of the parameters, they are modeled as log-linear functions:

$$v_k = v_k(\boldsymbol{x}) = \exp\left(\sum_{d=1}^{r} w_d^{(k)} \cdot x_d\right) = \exp\left(\left\langle \boldsymbol{w}^{(k)}, \boldsymbol{x} \right\rangle\right). \tag{11}$$

The parameters of the label ranking model, namely the $w_d^{(k)}$ $(1 \leq k \leq K,\ 1 \leq d \leq r)$, are estimated by maximum likelihood inference.

Given estimates of these parameters, prediction for new query instances $\boldsymbol{x}$ can be done in a straightforward way: $\hat{\boldsymbol{v}} = (\hat{v}_1, \ldots, \hat{v}_K)$ is computed based on (11), and a ranking $\hat{\pi}$ is determined by sorting the labels y_k in decreasing order of their (predicted) skills $\hat{v}_k$. This ranking $\hat{\pi}$ is a reasonable prediction, as it corresponds to the mode of the distribution $\mathbf{P}(\cdot \mid \hat{\boldsymbol{v}})$.

4.3 Plackett–Luce model with joint features

In (11), the skill of the label y_k is modeled as a log-linear function of $\boldsymbol{x}$, with a label-specific weight vector $\boldsymbol{w}^{(k)}$. In the context of dyad ranking, this approach can be generalized to the modeling of skills for dyads as follows:

$$v(\boldsymbol{z}) = v(\boldsymbol{x}, \boldsymbol{y}) = \exp\Big(\big\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \big\rangle\Big), \tag{12}$$

where Φ is a joint feature map (Tsochantaridis et al. 2005). Again, given an estimation $\hat{\boldsymbol{w}}$ of the parameter in (12), a ranking of a set of dyads

$$\varrho = \left\{\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(M)}\right\} \tag{13}$$

is predicted by sorting the dyads $\boldsymbol{z}^{(i)}$ in descending order of the corresponding skills $\hat{v}_i = v(\boldsymbol{z}^{(i)})$. This approach will be studied in more depth in the next two sections, for two different instantiations of the joint feature map.

5 Bilinear Plackett–Luce model (BilinPL)

As a first instantiation of the joint feature PL model (12), we consider the bilinear model (BilinPL) originally proposed by Schäfer and Hüllermeier (2015):

$$v(z) = v(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(\boldsymbol{x}^\top \mathbf{W} \boldsymbol{y}\right). \tag{14}$$

This model is obtained by defining Φ as the Kronecker (tensor) product:

$$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \otimes \boldsymbol{y} = \left(x_1 \cdot y_1, x_1 \cdot y_2, \ldots, x_r \cdot y_c\right) = \mathbf{vec}\left(\boldsymbol{x}\, \boldsymbol{y}^\top\right), \tag{15}$$

which is a vector of length $p = r \cdot c$ consisting of all pairwise products of the components of $\boldsymbol{x}$ and $\boldsymbol{y}$, also known as cross-products. Thus, the inner product $\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \rangle$ can be rewritten as a bilinear form $\boldsymbol{x}^\top \mathbf{W} \boldsymbol{y}$ with an $r \times c$ matrix $\mathbf{W} = (w_{i,j})$; the entry $w_{i,j}$ can be considered as the weight of the interaction term $x_i y_j$.

The bilinear model (14) appears to be the most obvious generalization of the linear model (11) in the context of dyad ranking. As such, it constitutes a natural starting point, although one may of course think of more complex, nonlinear extensions, for example using (pairwise) kernel functions (Basilico and Hofmann 2004). Likewise, one may also think of joint feature maps even simpler than the cross product $\boldsymbol{x} \otimes \boldsymbol{y}$, for example the concatenation $[\boldsymbol{x}, \boldsymbol{y}]$. This, however, would not allow for capturing interactions between the dyad members. Besides, it is not a proper generalization, as it does not cover (11) as a special case.

5.1 Identifiability of the bilinear PL model

The bilinear PL model introduced above defines a probability distribution on dyad rankings that is parameterized by the weight matrix $\mathbf{W}$. An important question, also from a learning point of view, concerns the identifiability of this model. Recall that, for a parameterized class of models $\mathcal{M}$, identifiability requires a bijective relationship between models $M_\theta \in \mathcal{M}$ and parameters θ, that is, models are uniquely identified by their parameters. Or, stated differently, parameters $\theta \neq \theta^*$ induce different models $M_\theta \neq M_{\theta^*}$. Identifiability is a prerequisite for a meaningful interpretation of parameters and, perhaps even more importantly, guarantees unique solutions for optimization procedures such as maximum likelihood estimation.

Obviously, the original PL model (7) with constant skill parameters $\boldsymbol{v} = (v_1, \ldots, v_K)$ is not identifiable, since the model is invariant against multiplication of the parameter by a constant factor $c > 0$: The models parameterized by $\boldsymbol{v}$ and $\boldsymbol{v}^* = (cv_1, \ldots, cv_K)$ represent exactly the same probability distribution, i.e., $\mathbf{P}(\pi \mid \boldsymbol{v}) = \mathbf{P}(\pi \mid \boldsymbol{v}^*)$ for all rankings π. The PL model is, however, indeed identifiable up to this kind of multiplicative scaling. Thus, by fixing one of the weights to the value 1, the remaining $K - 1$ weights can be uniquely identified.

Now, what about the identifiability of our bilinear PL model, i.e., to what extent is such a model uniquely identified by the parameter $\mathbf{W}$? We can show the following result (a proof is given in Appendix A).

Proposition 1 *Suppose the feature representation of labels does not include a constant feature, i.e., $|\mathcal{Y}_i| > 1$ for each of the domains in (2), and that the feature representation of instances includes at most one such feature (accounting for a bias, i.e., an intercept of the bilinear model). Then, the bilinear PL model with skill values defined according to (14) is identifiable.*

5.2 Comparison between the linear and bilinear PL model

It is not difficult to see that the linear model (11), subsequently referred to as LinPL, is indeed a special case of the bilinear model (14). In fact, the former is recovered from the latter by means of a (1-of-K) dummy encoding of the alternatives: The label y_k is encoded by a K-dimensional vector with a 1 in position k and 0 in all other positions. The columns of the matrix $\mathbf{W}$ are then given by the weight vectors $\boldsymbol{w}^{(k)}$ in (11).

The other way around, LinPL can also be applied in the setting of dyad ranking, provided the domain $\mathcal{Y}$ of the alternatives is finite. To this end, one would simply introduce one "meta-label" Y_k for each feature combination $(y_1, \ldots, y_c)$ in (2) and apply a standard label ranking method to the set of these meta-labels.[1] Therefore, both approaches are in principle equally expressive. Still, an obvious problem of this transformation is the potential size[2]

$$K = |\mathcal{Y}| = |\mathcal{Y}_1| \times |\mathcal{Y}_2| \times \cdots \times |\mathcal{Y}_c|$$

of the label set thus produced, which might be huge. In fact, the number of parameters that need to be learned for the model (11) is $r \cdot |\mathcal{Y}|$, i.e., $r \cdot a^c$ under the assumptions that each feature has a values. For comparison, the number of parameters is only $r \cdot c$ in the bilinear model. Moreover, all information about relationships between the alternatives (such as shared features or similarities) are lost, since a standard label ranker will only use the name of a meta-label while ignoring its properties.

Against the background of these considerations, one should expect dyad ranking to be advantageous to standard label ranking provided the assumptions underlying the bilinear model (14) are indeed valid, at least approximately. In that case, learning with (meta-)labels and disregarding properties of the alternatives would come with an unnecessary loss of information (that would need to be compensated by additional training data). In particular, using the standard label ranking approach is supposedly problematic in the case of many meta-labels and comparatively small amounts of training data.

Having said that, dyad ranking could be problematic if the model (14) is in fact a misspecification: If the features are not meaningful, or the bilinear model is not properly reflecting their interaction, then learning on the basis of (14) cannot be successful. The main observations can thus be summarized as follows:

- The linear PL model, like standard label ranking in general, assumes all alternatives to be known beforehand and to be included in the training process. If generalization beyond alternatives encountered in the training process is needed, then BilinPL can be used while LinPL cannot.
- If the assumption (14) of the bilinear model is correct, then BilinPL should learn faster than LinPL, as it needs to estimate fewer parameters. Yet, since LinPL can represent all dependencies that can be represented by BilinPL, the learning curve of the former should reach the one of the latter with growing sample size.

[1] This approach could be compared to the reduction of multi-label to multi-class classification via the label powerset transformation (Tsoumakas and Katakis 2007).

[2] This is an upper bound, since in practice, not all feature combinations are necessarily realized.

5.3 Learning the bilinear PL model

5.3.1 Learning via maximum likelihood estimation

Suppose training data $\mathcal{D}$ to be given in the form of a set of rankings (4), i.e., rankings $\rho_1, \ldots, \rho_N$ of the following kind:

$$\rho_n : \left(\boldsymbol{x}_n^{(1)}, \boldsymbol{y}_n^{(1)}\right) \succ \left(\boldsymbol{x}_n^{(2)}, \boldsymbol{y}_n^{(2)}\right) \succ \cdots \succ \left(\boldsymbol{x}_n^{(M_n)}, \boldsymbol{y}_n^{(M_n)}\right). \tag{16}$$

The likelihood of the parameter vector $\boldsymbol{w}$ is then given by

$$\mathcal{L}_n(\boldsymbol{w}; \mathcal{D}) = \mathbf{P}(\mathcal{D} \mid \boldsymbol{w}) = \prod_{n=1}^{N} \prod_{m=1}^{M_n} \frac{\exp\left(\boldsymbol{w}^\top \boldsymbol{z}^{(n,m)}\right)}{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \boldsymbol{z}^{(n,l)}\right)},$$

where we set $\boldsymbol{z}^{(n,m)} = \boldsymbol{x}_n^{(m)} \otimes \boldsymbol{y}_n^{(m)}$. Like in the case of the linear PL model, the learning problem can now be formalized as finding the maximum likelihood (ML) estimate, i.e., the parameter

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\ \mathcal{L}(\boldsymbol{w}; \mathcal{D}), \tag{17}$$

that maximizes the likelihood or, equivalently, minimizes the negative log-likelihood (NLL)

$$\ell_n(\boldsymbol{w}; \mathcal{D}) = -\sum_{n=1}^{N} \sum_{m=1}^{M_n} \boldsymbol{w}^\top \boldsymbol{z}^{(n,m)} + \sum_{n=1}^{N} \sum_{m=1}^{M_n} \log\left(\sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \boldsymbol{z}^{(n,l)}\right)\right). \tag{18}$$

Since there is no analytical solution to this optimization problem, one has to rely on gradient-based methods, where the gradient $\nabla \ell_n = \left(\frac{\partial \ell_n}{\partial w_1}, \ldots, \frac{\partial \ell_n}{\partial w_p}\right)^\top$ is defined by the partial derivatives

$$\frac{\partial \ell_n}{\partial \boldsymbol{w}_i} = \sum_{n=1}^{N} \sum_{m=1}^{M_n - 1} g(\boldsymbol{w})^{-1} h(\boldsymbol{w}) - \sum_{n=1}^{N} \sum_{m=1}^{M_n - 1} z_i^{(n,m)}, \tag{19}$$

with

$$g(\boldsymbol{w}) = \sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \boldsymbol{z}^{(n,l)}\right) \text{ and}$$

$$h(\boldsymbol{w}) = \sum_{l=m}^{M_n} z_i^{(n,l)} \exp\left(\boldsymbol{w}^\top \boldsymbol{z}^{(n,l)}\right).$$

For such methods, the convexity of the function to be minimized is of critical importance (Boyd and Vandenberghe 2004). In Appendix B, we therefore show that the negative log-likelihood function (18) is convex.

5.3.2 Optimization via iterative majorization

The MM algorithm proposed by Hunter (2004) belongs to the standard approaches for maximum likelihood estimation of the basic PL model parameters $\boldsymbol{v}$. The acronym MM stands for majorization-minorization (minorization-majorization) in the case of minimization (maximization) problems. The prominent EM algorithm can be seen as a special instance of MM (Dempster et al. 1977). While the standard Newton algorithm could in principle be applied to minimize the convex PL objective, it is known that the MM approach is more robust and

faster in direct comparison (Guiver and Snelson 2009). Although MM algorithms often need more iterations, they perform specifically well when operating far from the optimum point (Lange et al. 2000). The MM algorithm is furthermore superior to Newton's method in the context of PL model training, because the latter requires special countermeasures to prevent erratic behavior (Hunter 2004). The overall advantage of MM over Newton in terms of speed is mainly due to the time-intensive inversion of the Hessian in any Newton step.

Recently proposed alternative optimization approaches are based on Bayesian inference. For example, Guiver and Snelson (2009) propose approximate inference based on expectation propagation and Caron and Doucet (2012) make use of Gibbs sampling. Maystre and Grossglauser (2015) propose an alternative approach for the basic PL model that is based on interpreting the maximum likelihood estimate as a stationary distribution of a Markov chain. This enables a fast and efficient spectral inference algorithm.

Recall that, in contrast to the basic PL model (7), the parameters v_i are not real numbers in the BilinPL model (14) but functions of a weight vector $\boldsymbol{w}$ and a joint-feature vector. To adopt the MM algorithm for obtaining $\boldsymbol{w}^*$, we take a closer look at MM and adopt the perspective of minimization by majorization of the NLL (18).

The MM algorithm is not a concrete method but rather a prescription for constructing optimization algorithms. The idea is to construct a surrogate (or auxiliary) function g that majorizes a particular objective function f. The surrogate function should be simpler than the original function and should "touch" it at the so-called supporting point $\boldsymbol{u}$, i.e., $g(\boldsymbol{w}, \boldsymbol{u}) \geq f(\boldsymbol{w})$ for all $\boldsymbol{w}$. The iterative majorization approach essentially consists of the following steps, which drive the value of the objective downhill (Borg and Groenen 2005; De Leeuw and Mair 2009):

1. Initialize the first supporting point $\boldsymbol{u} = \boldsymbol{u_0}$.
2. Find update $\boldsymbol{w}$, so that $g(\boldsymbol{w}, \boldsymbol{u}) \leq g(\boldsymbol{u}, \boldsymbol{u})$.
3. If $f(\boldsymbol{u}) - f(\boldsymbol{w}) < \epsilon$, then stop and return $\boldsymbol{w}$, else set $\boldsymbol{u} = \boldsymbol{w}$ and go back to line 2

For the case of BilinPL, we define the function $f(\boldsymbol{w}) = \ell_n(\mathcal{D}, \boldsymbol{w})$. To get rid of the logarithm in (18) and to simplify the objective, one can exploit the concavity of the logarithm, i.e.,

$$\ln(x) \leq \ln(y) + \frac{x}{y} - 1, \tag{20}$$

which is a consequence of the "supporting hyperplane inequality" (Lange 2016) or first-order condition of concavity (Boyd and Vandenberghe 2004):

$$f(y) \leq f(x) + f'(x)(y - x), \tag{21}$$

where the right-hand side of (21) is the first-order Taylor approximation of f at the point x. With (20), it is possible to express the surrogate function $g(\boldsymbol{w}, \boldsymbol{u})$ as follows:

$$\begin{aligned} g(\boldsymbol{w}, \boldsymbol{u}) = &\sum_{n=1}^{N} \sum_{m=1}^{M_n} \left(\frac{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \boldsymbol{z}^{(n,l)}\right)}{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{u}^\top \boldsymbol{z}^{(n,l)}\right)} \right) \\ &+ \sum_{n=1}^{N} \sum_{m=1}^{M_n} \log \left(\sum_{l=m}^{M_n} \exp\left(\boldsymbol{u}^\top \boldsymbol{z}^{(n,l)}\right) \right) \\ &- \sum_{n=1}^{N} \sum_{m=1}^{M_n} 1 - \sum_{n=1}^{N} \sum_{m=1}^{M_n} \boldsymbol{w}^\top \boldsymbol{z}^{(n,m)}. \end{aligned} \tag{22}$$

Its gradient with respect to $\boldsymbol{w}$ is given by the partial derivatives

$$\frac{\partial g}{\partial \boldsymbol{w}_i} = \sum_{n=1}^{N} \sum_{m=1}^{M_n} \left(\frac{\sum_{l=m}^{M_n} \boldsymbol{z}_i^{(n,l)} \exp\left(\boldsymbol{w}^\top \boldsymbol{z}^{(n,l)}\right)}{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{u}^\top \boldsymbol{z}^{(n,l)}\right)} \right) - \sum_{n=1}^{N} \sum_{m=1}^{M_n} \boldsymbol{z}_i^{(n,m)}, \tag{23}$$

$1 \leq i \leq d$. To realize step 2 of the MM algorithm, i.e., the update step, one needs to find a vector $\boldsymbol{w}^*$ such that, at least approximately, $\nabla g \approx 0$ (for a fixed vector $\boldsymbol{u}$). This can be accomplished by performing a single Newton step (Lange et al. 2000) with

$$\boldsymbol{w} = \boldsymbol{u} - [Hg(\boldsymbol{u}, \boldsymbol{u})]^{-1} \nabla f(\boldsymbol{u}). \tag{24}$$

The Hessian of g is computationally less complex than that of f. Yet, for the minimization of (18), it turns out that a quasi-Newton approach such as the L-BFGS method (Liu and Nocedal 1989) is in practice much more efficient. The proclaimed advantages of operating well in regions far from the optimum and the prevention of erratic behavior could not be observed with L-BFGS in the experiments provided in Sect. 8.

5.3.3 Online learning using stochastic gradient descent

An interesting alternative approach, especially in the large-scale learning regime, is stochastic gradient descent (SGD) with selective sampling (Bottou 1998, 2010). The core idea of SGD is to update model parameters iteratively on the basis of a randomly picked example. The algorithm is explicated for the case of dyad ranking with the BilinPL model in Algorithm 1. It requires the specification of a data source, the total number of iterations, an initial learning rate, and a regularization parameter.

The probabilistic nature of BilinPL can be leveraged (in line 3 of the algorithm) for sampling training instances in a selective way. More specifically, for each randomly sampled instance, our model allows for deriving the probability of the corresponding dyad ranking (given the current parameter estimates), as well as the probability of alternative rankings. Thus, it is possible quantify the uncertainty of the (current) model in its prediction and, therefore, to implement a kind of active learning via uncertainty sampling.

Algorithm 1 BilinPL with SGD

Require: data set $\mathcal{S}$, initial learn rate η_0, regularization parameter λ, no. iterations
1: $i \leftarrow 0$
2: **for** $i <$ `nIter` **do**
3: $\quad \rho \leftarrow$ **sample a dyad ranking** $(\mathcal{S})$
4: $\quad \boldsymbol{w} \leftarrow$ **update**$(i, \eta_0, \lambda, \rho)$
5: $\quad i \leftarrow i + 1$
6: **end for**
7: **return** $\boldsymbol{w}$

6 Plackett–Luce networks

Due to the bilinearity assumption, BilinPL comes with a relatively strong bias. This may or may not turn out as an advantage, depending on whether the assumption holds sufficiently well, but in any case requires a proper feature engineering. The approach introduced in this

section, called Plackett–Luce Network (PLNet), offers an alternative: It allows for *learning* a highly nonlinear joint-feature map expressed in terms of a neural network (Schäfer and Hüllermeier 2016).

6.1 Architecture

The core idea of PLNet is to learn real-valued (latent) utility functions $u = g(\boldsymbol{x}, \boldsymbol{y})$, where g is the function implemented by a single multi-layer feed-forward neural network, and u determines the strength of the dyad $z = (\boldsymbol{x}, \boldsymbol{y})$ in a PL model. More specifically, the probability of observing the ranking

$$\rho : (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}) \succ (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}) \succ \cdots \succ (\boldsymbol{x}^{(M)}, \boldsymbol{y}^{(M)})$$

is given by

$$\mathbf{P}(\rho \mid \boldsymbol{v}) = \prod_{m=1}^{M-1} \frac{v_m}{\sum_{l=m}^{M} v_l} = \prod_{m=1}^{M-1} \frac{\exp\left(u^{(m)}\right)}{\sum_{l=m}^{M} \exp\left(u^{(l)}\right)}, \tag{25}$$

that is, by the PL probabilities induced by the parameters

$$v_m = \exp\left(u^{(m)}\right) = \exp\left(g(\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\right).$$

PLNet is a multi-layer perceptron (MLP), the structure of which is shown in Fig. 2. It consists of multiple layers and takes as input two vectors corresponding to the members of a dyad. There is at least one hidden layer with nodes using a sigmoidal activation function. The output layer produces a scalar value of the form $u = \langle \boldsymbol{w}^L, \boldsymbol{a}^{L-1} \rangle + b^L$. Technically, a bias term b^L is not needed in the last layer, as it has no effect on the PLNet model (25): With the choice of the exponential function (to ensure positivity of the PL parameters), and the independence of b^L of the inputs $\boldsymbol{x}$ and $\boldsymbol{y}$, we have

$$v = \exp\left(\langle w^L, a^{L-1} \rangle + b^L\right) = \exp(b^L) \cdot \exp\left(\langle w^L, a^{L-1} \rangle\right), \tag{26}$$

and as noted before, the PL model is invariant against multiplication of the weights with a positive scalar.

By the special choice of the linear activation function for the neuron at the output layer, the architecture is related to the joint-feature PL reference model (12) outlined in Sect. 4.3. The activation output of the neurons of the penultimate layer can be considered as a joint-feature vector $\Phi(\boldsymbol{x}, \boldsymbol{y})$, which is then linearly combined with a vector of weights to produce a utility score. Thus, the upper part of the network, i.e., the input layer down to the penultimate layer, can be considered as a joint-feature map $\Phi(\boldsymbol{x}, \boldsymbol{y})$.

6.2 Training

Our training procedure builds on the basic idea of back-propagation (Werbos 1974; Rumelhart et al. 1986), which, however, needs to be modified in various ways. First, the original back-propagation algorithm is derived for the squared error between prediction and target as a loss function, whereas PLNet seeks to minimize the negative log-likelihood. Second, the targets in PLNet are rankings instead of real numbers.

The processing of a single ranking

$$\rho_n : \left(\boldsymbol{x}_n^{(1)}, \boldsymbol{y}_n^{(1)}\right) \succ \left(\boldsymbol{x}_n^{(2)}, \boldsymbol{y}_n^{(2)}\right) \succ \cdots \succ \left(\boldsymbol{x}_n^{(M_n)}, \boldsymbol{y}_n^{(M_n)}\right)$$

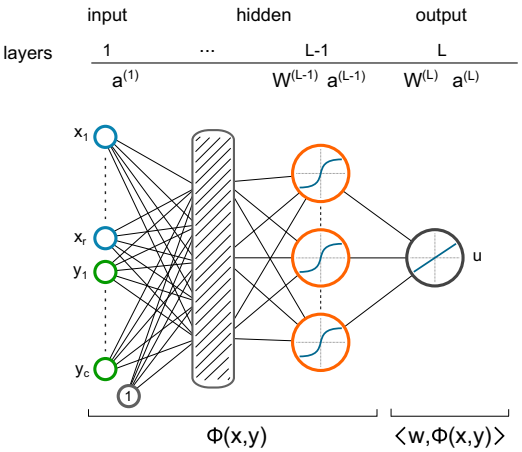

Fig. 2 PLNet architecture. This kind of feed-forward neural network is composed of several layers. Dyad members $z = (\boldsymbol{x}, \boldsymbol{y})$ are entered at the input layer, whose nodes are connected with nodes of the next layer. Inner (hidden) layers have nodes endowed with a non-linear activation function. The output layer in contrast only consists of a single node, which is equipped with the identity activation function

Algorithm 2 PLNet Training Step

Require: Training example $\rho_n : (\boldsymbol{x}_n^{(1)}, \boldsymbol{y}_n^{(1)}) \succ (\boldsymbol{x}_n^{(2)}, \boldsymbol{y}_n^{(2)}) \succ \cdots \succ (\boldsymbol{x}_n^{(M_n)}, \boldsymbol{y}_n^{(M_n)})$

1: Create M_n copies of master network g_t.

2: Enter $\left(\boldsymbol{x}_n^{(k)}, \boldsymbol{y}_n^{(k)}\right)$ into network $g_{t,k}$, $1 \leq k \leq M_n$.

3: Calculate derivatives using the outputs $\theta = \left\{u^{(1)}, \ldots, u^{(M_n)}\right\}$.

4: Evaluate $\Delta w_{j,i}^{(k)}$ on networks $g_{t,k}$

5: $g_{t+1} \leftarrow \text{update}\left(g_t, \sum_k \Delta w_{j,i}^{(k)}\right)$

is described in Algorithm 2. In training step t, the current PL network g_t, which we refer to as the *master* network, is cloned M_n times, yielding M_n (read-only) PL networks $g_{t,k}$. The kth dyad of ρ_n, when entered into the network $g_{t,k}$, yields the output $u^{(k)}$. Proceeding that way for all $1 \leq k \leq M_n$, it is possible to evaluate the probability of that ranking according to (25). Moreover, we can calculate errors to be used for back-propagation. Recall that the NLL is given by

$$E_n = -\log \mathbf{P}\left(\rho_n \mid \boldsymbol{v}\right) = \sum_{m=1}^{M_n-1} \log \sum_{l=m}^{M_n} \exp\left(u^{(l)}\right) - \sum_{m=1}^{M_n-1} u^{(m)}. \tag{27}$$

With this specification of the NLL, the output error of the kth network, $1 \leq k \leq M_n$, can be expressed as follows:

$$\delta_L = \frac{\partial E_n}{\partial u^{(k)}} = \left[\sum_{m=1}^{k} \frac{\exp\left(u^{(k)}\right)}{\sum_{l=m}^{M_n} \exp\left(u^{(l)}\right)}\right] - 1. \tag{28}$$

This value can then be used for back-propagation on the kth virtual network to calculate $\frac{\partial E_n}{\partial w_{j,i}}$, and thus $\Delta w_{j,i} = \eta \frac{\partial E_n}{\partial w_{j,i}}$, with the learning rate η. The updates of the weights are not realized on the kth network directly, but they are used to carry out the weight adjustments on the *master* network instead. This can either be accomplished by a single update

$$w_{j,i} \leftarrow w_{j,i} - \sum_k \Delta w_{j,i}^{(k)}, \tag{29}$$

or by a sequence of updates $w_{j,i} \leftarrow w_{j,i} - \Delta w_{j,i}^{(k)}$, $1 \leq k \leq M_n$. To justify the accumulation of the weight changes from the individual networks, consider the error term (27) as a function of the network outputs, i.e.,

$$E_n\left(u^{(1)}(\theta), u^{(2)}(\theta), \ldots, u^{(M_n)}(\theta)\right). \tag{30}$$

Note that E_n depends on the network parameters θ only *indirectly* via the outputs. These can in turn be considered as functions that share the same variables. Taking the total derivative of (30) with respect to $w_{j,i} \in \theta$, we obtain

$$\frac{\partial E_n}{\partial w_{j,i}} = \frac{\partial E_n}{\partial u^{(1)}} \frac{\partial u^{(1)}}{\partial w_{j,i}} + \frac{\partial E_n}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial w_{j,i}} + \cdots + \frac{\partial E_n}{\partial u^{(M_n)}} \frac{\partial u^{(M_n)}}{\partial w_{j,i}}. \tag{31}$$

A summand $\frac{\partial E_n}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial w_{j,i}}$ in (31) coincides with the objective of back-propagation on a single network k, i.e., the evaluation of $\frac{\partial E_n}{\partial w_{j,i}}$ with respect to the kth network's output $u^{(k)}$.

After an update has taken place, the procedure can be repeated on the remaining training examples. The algorithm stops when the error has been sufficiently diminished. We call this procedure, which is a hybrid between the online and the batch variant of back-propagation, *staged* back-propagation (SBP). In batch training, the updated weights are accumulated for n training examples before updating the network, whereas in online back-propagation, the network is updated after each example. In SBP, the network is updated in an online manner over the training examples, but in a batch-wise mode on the elements of each single example.

We suggest *early stopping* as a means for regularization to prevent overfitting.[3] To this end, we track the NLL values of the training and validation data during the learning process (Prechelt 2012). A good point to stop the training and to prevent over-fitting is when the validation NLL values start to increase again. Finally, predictions can be carried out straightforwardly using a trained PLNet. Given a set of dyads, we simply need to sort them in descending order of their utilities—as already explained, the ranking thus obtained corresponds to the mode of the PL distribution.

6.3 Applicability

PLNet can be applied to dyad ranking problems as well as to label ranking problems. As for the latter, one possibility is to use the dummy encoding mentioned in Sect. 5.2, in which the vectors of the domain $\mathcal{Y}$ are expressed as one-hot vectors. As in this case all values y_i in the input layer will be 0, except one having the value 1, this approach could also be implemented by omitting the input vector $\boldsymbol{y}$ altogether, and instead adding label-specific biases for the activation functions of the first hidden layer neurons.

By relaxing the input type from a vector pair $(\boldsymbol{x}, \boldsymbol{y})$ to a single input vector $\boldsymbol{z}$, PLNet would be applicable on the problem of object ranking, too (Cohen et al. 1999; Kamishima et al. 2011). It would consequently be possible to apply PLNet also on ordered joint query-document vectors $\boldsymbol{z} = \Psi(q, d)$ as commonly encountered in the *learning to rank* domain.

6.4 Comparison with existing NN-based approaches

In this section, we provide a brief overview of other (preference) learning methods based on neural networks that share some similarities with PLNet.

[3] Of course, other techniques could be used as well, including standard L2 regularisation. Since early stopping works well, a thorough comparison of different alternatives has not yet been done.

Comparison training refers to a framework introduced for learning from pairwise preferences with a neural network (Tesauro 1989). The network architecture consists of two subnetworks, which are connected to a single output node that indicates which of the two inputs is preferred over the other. The weights associated with the last hidden layer of one subnetwork is the mirrored version of the other subnetwork's weights. Two principal problems are addressed with this setup, namely efficiency and consistency. In the evaluation phase, only a single subnetwork is used to evaluate n alternatives, which are then sorted according to their scores. The network essentially implements a (latent) utility function (thereby enforcing transitivity of the predicted preferences).

A similar approach called SortNet is proposed by Rigutini et al. (2011). They introduce a three-layered network architecture called CmpNN, which takes two input object vectors and outputs two real-valued numbers. The architecture establishes a preference function, for which the properties of reflexivity and symmetry (but not transitivity) are ensured by a weight-sharing technique. Huybrechts (2016) uses the CmpNN architecture (with 3 hidden layers) in the context of document retrieval.

The Bradley–Terry model was re-parameterized by a single layer neural network (consisting of a single sigmoid node) by Menke and Martinez (2008). It is used for predicting the outcome of two-team group competitions by rating individuals in the application of e-sports. The model offers several extensions, such as home-field advantage, player contributions, time effects, and uncertainty. This model differs from our approach in the sense of being tailored for pairwise group comparisons, albeit considering individual skills. The inclusion of feature vectors is of no concern in this model, and the extension to rankings is only suggested for future work.

The label ranking problem, introduced in Sect. 2.1, has been tackled with neural network approaches previously (Ribeiro et al. 2012; Kanda et al. 2012). A multi-layer perceptron (MLP) has been utilized by Kanda et al. (2012) to produce recommendations on meta-heuristics (labels) on different meta-features (instances) within the realm of meta-learning. This kind of neural network exhibits an output layer with as many nodes as there are labels. The error signal used to modify the network's weights is formed by using the mean squared error on the target rank positions of the labels. In Ribeiro et al. (2012) more effort has been spent to incorporate label ranking loss information into the back-propagation procedure. To this end some variations of this procedure have been investigated. Both architectures are similar to each other and have two essential limitations: first, they depend on a fixed number of labels, and second, they cannot cope with incomplete ranking observations. In addition, they lack the ability to provide probabilistic information on their predictions. Zhang and Zhou (2006) train neural networks to minimize the ranking loss in multi-label classification. Thus, the network is adjusted such that, given an instance as an input, higher scores are produced as output for relevant (positive) labels and lower scores for irrelevant (negative) labels.

In the domain of information retrieval, the neural network-based approaches RankNet and ListNet have a probabilistic foundation (Burges et al. 2005; Cao et al. 2007). RankNet (Burges et al. 2005) uses pairwise inputs to learn a utility scoring function with the cross entropy loss. To this end, the training data consists of sample pairs together with target probabilities. These quantify the probability to rank the first sample higher than the second. With the introduction of target probabilities, this approach enables the interesting possibility of modeling ties between samples. ListNet (Cao et al. 2007; Luo et al. 2015) similarly uses the cross entropy as a metric, but in contrast to RankNet it processes lists of samples instead of pairwise preferences as basic observation. There are, however, some important differences between ListNet and our approach:

– The learning approach in ListNet addresses only a special case of the PL distribution, namely the case of top-k data with $k = 1$.
– In ListNet, a *linear* neural network is used. This is in contrast to our approach, in which non-linear relationships between inputs and outputs are learned. Linearity in the ListNet approach implies that much emphasis must be put on engineering joint feature input vectors.
– In ListNet, the query-document features are associated with absolute scores (relevance degrees) as training information, i.e., quantitative data, whereas PLNet deals with rankings, i.e., data of qualitative nature.[4]

7 Multidimensional unfolding of dyad ranking models

Multidimensional unfolding is an extension of multidimensional scaling (MDS) for two-way preferential choice data (Borg and Groenen 2005). As such, it is a useful visualization tool for analyzing and discovering preference structures. In this section, we examine how dyad ranking data can be visualized by combining the learned models with multidimensional unfolding.

7.1 The unfolding problem

In multidimensional unfolding, the data is given as preference scores (e.g., rank-orders of preferences), typically of n_1 individuals for a set of n_2 items (Borg et al. 2012). Individuals are represented as "ideal points" in a common lower-dimensional space together with the items, in such a way that their distances to the items match with their preference scores. Unfolding methods produce point configurations in a lower-dimensional space as illustrated in Fig. 3. The main assumption in unfolding is that all individuals perceive the world in the same way, which is reflected by one fixed configuration of the items. The differences between the individuals' preferences are realized by different ideal point positions relative to the item points (Borg and Groenen 2005). Early works on unfolding by Coombs (1950) were motivated by the idea of expressing preferences of individuals over items on a joint continuum called J scale, i.e., a line. Folding the line on an individual's point allows one to study the preferences of that individual by inspecting the resulting locations of the objects on it. The term "unfolding" then refers to the inverse process of finding the common scale from given preferences (Busing 2010).

Technically, unfolding is a kind of multidimensional scaling where the within-sets proximities are missing. The objective in MDS is to find a point configuration $\boldsymbol{X}$ in a lower-dimensional space, such that the point distances are in accordance with the distances among the original data points. Kruskal (1964) introduced the squared error function (raw) stress,[5] which is the objective that needs to be minimized:

$$\sigma_{\text{raw}}(\boldsymbol{X}) = \sqrt{\sum_{(i,j)} \left(\hat{d}_{i,j} - d_{i,j}(\boldsymbol{X})\right)^2}, \tag{32}$$

[4] We argue that using query-document-associated scores as PL model parameters is anyway questionable, especially because these such scores are normally taken from an ordinal scale.

[5] Stress is an acronym for standardized residual sum-of-squares.

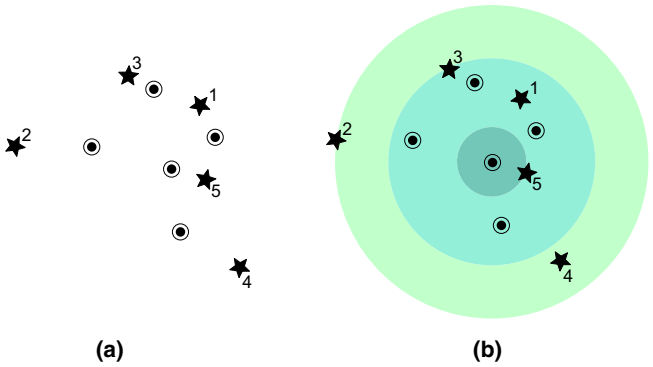

Fig. 3 Illustration of multidimensional unfolding. **a** An unfolding solution, which is a configuration of points: users are represented as dots and items as stars. In **b** isocontour circles of a particular user are added. They facilitate reading his preference for the items. Here, the preference order is $i_5 \succ i_1 \succ i_3 \succ i_4 \succ i_2$

where $d_{i,j}(\boldsymbol{X})$ are Euclidean distances between points $\boldsymbol{x}_i, \boldsymbol{x}_j \in \boldsymbol{X}$ in the lower-dimensional MDS space, and with

$$\hat{d}_{i,j} = f(\delta_{i,j}), \tag{33}$$

the so-called disparities. Depending on the type of transformation $f(\cdot)$ on the input dissimilarities $\delta_{i,j}$, one distinguishes between *nonmetric* (or ordinal) MDS and *metric* MDS.

Instead of a single configuration matrix $\boldsymbol{X}$, there are two matrices involved in unfolding: $\boldsymbol{X}_1$ of dimension $n_1 \times p$ for the configuration of individuals, and $\boldsymbol{X}_2$ of dimension $n_2 \times p$ for the items configuration, where p denotes the dimensionality of the unfolding space. Correspondingly, the problem in unfolding can be stated as minimizing the following (raw) stress function:

$$\sigma_r^2(\boldsymbol{X}_1, \boldsymbol{X}_2) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left(\hat{d}_{i,j} - d_{i,j}(\boldsymbol{X}_1, \boldsymbol{X}_2)\right)^2. \tag{34}$$

7.2 Dyadic unfolding

With a trained dyad ranking model, it is possible to produce a matrix of skills for pairs of possibly new feature vectors from the domains $\mathcal{X}$ and $\mathcal{Y}$. Let $\boldsymbol{X}$ be an $n_1 \times d_1$ matrix and $\boldsymbol{Y}$ an $n_2 \times d_2$ matrix of feature vectors. Then, the skills $v_{i,j}$ produced by the model on all pairwise vectors of $\boldsymbol{X}$ and $\boldsymbol{Y}$ can be grouped into a matrix $\boldsymbol{S}$ of dimensionality $n_1 \times n_2$.

A reasonable goal for visualizing this data would be to find a lower-dimensional representation that takes all available proximities into account, which are in this case dissimilarities Δ_X between objects (feature vectors) from $\mathcal{X}$, dissimilarities Δ_Y between objects (feature vectors) from $\mathcal{Y}$, and the preferences on dyads, which can be represented by turning their (estimated) skills into dissimilarities Δ_S. What is sought is a low-dimensional configuration of points $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, such that distances of points within $\boldsymbol{X}_1$ are in accordance with the dissimilarities Δ_X, distances of points within $\boldsymbol{X}_2$ are in accordance with the dissimilarities Δ_Y, and distances of points between $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are in accordance with Δ_S. This objective can be expressed as

$$\sigma_D = s(\Delta_X, \Delta_Y, \Delta_S, \boldsymbol{X}_1, \boldsymbol{X}_2). \tag{35}$$

The strength of the relation between the proximities and the resulting configurations could for example be realized as a weighted sum of stress terms

$$\sigma_D^2 = \alpha\sigma_X^2 + \beta\sigma_Y^2 + \gamma\sigma_S^2, \tag{36}$$

with $\alpha + \beta + \gamma = 1$. Expanding (36) yields

$$\sigma_D^2 = \alpha \left[\sum_{(i,j)} \left(\hat{d}_{i,j}^{[X]} - d_{i,j}(\boldsymbol{X}_1) \right)^2 \right] + \beta \left[\sum_{(i,j)} \left(\hat{d}_{i,j}^{[Y]} - d_{i,j}(\boldsymbol{X}_2) \right)^2 \right] + \gamma \left[\sum_{(i,j)} \left(\hat{d}_{i,j}^{[S]} - d_{i,j}(\boldsymbol{X}_1, \boldsymbol{X}_2) \right)^2 \right]. \tag{37}$$

Typical formulations of stress such as (32), like the one used in the formulation (37), require dissimilarities $\delta_{i,j}$ or dissimilarity transformations called disparities $\hat{d}_{i,j} = f(\delta_{i,j})$. For the first two terms, these disparities can easily be obtained as pairwise Euclidean distances between the feature vectors within $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively.

For the transformation of the skills $v_{i,j}$, there are several possibilities. Borg (1981) discusses functional relationships between the value scale v and distances in an unfolding model. The most obvious relationship is $d_{i,j} = 1/v_{i,j}$, which we call transformation $t_1(v_{i,j}) = 1/v_{i,j}$. Borg points out that the major drawback of this formula is that for almost similar items it requires v to become infinitely large. A better alternative, originally also motivated in (Luce 1961; Krantz 1967), is $t_2(v_{i,j}) = d_{i,j} = \log(1/v_{i,j}) = -\log(v_{i,j})$.[6] Another transformation (t_3) is the rank-transformation, which creates rank numbers in descending order of the skill values.

Combining unfolding with the PL models offers the possibility to enrich the visualizations with probabilistic information. For example, consider the neighborhood of a dyad member $\boldsymbol{x}_i$ (or ideal point in the unfolding terminology), and suppose this neighborhood consists of M dyad members $\boldsymbol{y}_j$. Given the corresponding skills $v(\boldsymbol{x}_i, \boldsymbol{y}_j)$, the complete distribution over the rankings of these M members can be determined. Moreover, for each $\boldsymbol{y}_j$, an interesting piece of information is its (marginal) distribution over the ranks $1 \leq k \leq M$, with

$$M(j,k) = \sum_{\pi\,:\,\pi(k)=j} \mathbf{P}(\pi \mid \boldsymbol{v}), \tag{38}$$

the probability that $\boldsymbol{y}_j$ is put on position k in the ranking. Annotating $\boldsymbol{y}_j$ with this distribution (or a part of it, e.g., the most probable positions) provides useful additional information.

7.2.1 Dyadic unfolding with SMACOF

The problem of minimizing (36) can be tackled by considering two facts on unfolding reported in the literature (Borg and Groenen 2005). Given the preferences of individuals on items as a matrix of dissimilarities Δ, it is possible to create an unfolding solution using any regular MDS method that supports missing values. To this end, the method is applied to the matrix in which between-set dissimilarities are given and within-set dissimilarities are missing. A matrix of that kind would exhibit a block structure as follows:

$$\Delta_{\text{MDS}} = \begin{bmatrix} - & \Delta \\ \Delta^\top & - \end{bmatrix}.$$

[6] In our models, the $v_{i,j}$ are ensured to be positive but not necessarily bounded. Therefore, we extend the mapping t_2 by a subsequent affine linear transformation to the unit interval.

A particular method that supports missing values and thus unfolding is SMACOF, which stands for "Stress Majorization of a Complicated Function" (De Leeuw and Heiser 1977; De Leeuw 1977). This is a multidimensional scaling technique that is based on iterative majorization. The "complicated" goal function in SMACOF defines the optimization problem and is a weighted sum of the squared error function called (raw) stress,

$$\sigma_r^2 = \sigma_r^2(\boldsymbol{X}) = \sum_{i<j} w_{i,j} \left(\hat{d}_{i,j} - d_{i,j}(\boldsymbol{X})\right)^2. \tag{39}$$

The non-negative weights $w_{i,j}$ in (39) were originally included and suggested by De Leeuw to provide more flexibility. They can be used to express the importance of the residuals $\hat{d}_{i,j} - d_{i,j}(\boldsymbol{X})$ or can be used to handle missing data (Groenen and van de Velden 2016). For *multidimensional unfolding*, the configuration matrix $\boldsymbol{X}$ can be decomposed in two matrices $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, which are of dimensionality $n_1 \times p$ and $n_2 \times p$, respectively. A weight matrix $\boldsymbol{W}$ can also be included in this case, which enables the aforementioned importance weighting or the indication of missing values. SMACOF requires $\boldsymbol{W}$ to be irreducible, symmetric, non-negative, and hollow. The matrix $\boldsymbol{W}$ is structured into four blocks and contains $(n_1 + n_2)^2$ many entries:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_{11} & \boldsymbol{W}_{12} \\ \boldsymbol{W}_{21} & \boldsymbol{W}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{W}_{12} \\ \boldsymbol{W}_{21} & \boldsymbol{0} \end{bmatrix},$$

where $\boldsymbol{W}_{11} = \boldsymbol{W}_{22} = 0$ accounts for the property of missing within-set proximities.

As reported in Borg and Groenen (2005, Chapter 11.3), weights like those in the weighted stress function (39) provide a certain degree of flexibility. It is for example possible to mimic other stress functions, such as those used in Sammon's mapping, elastic scaling, or S-Stress. Moreover, weights can be used to encode reliability on a proximity, so that proximities with large weights have more impact on a resulting MDS solution than those that are less reliable.

Combining all this, "dyadic" unfolding can be performed by specifying an $(n_1 + n_2)^2$ weight matrix $\boldsymbol{W}$ as

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_{11} & \boldsymbol{W}_{12} \\ \boldsymbol{W}_{21} & \boldsymbol{W}_{22} \end{bmatrix} = \begin{bmatrix} \alpha \boldsymbol{1}\boldsymbol{1}^\top & \gamma \boldsymbol{1}\boldsymbol{1}^\top \\ (\gamma \boldsymbol{1}\boldsymbol{1}^\top)^\top & \beta \boldsymbol{1}\boldsymbol{1}^\top \end{bmatrix}. \tag{40}$$

The dissimilarity matrix has a block structure and is given by

$$\Delta = \begin{bmatrix} \Delta_X & \Delta_S \\ \Delta_S^\top & \Delta_Y \end{bmatrix},$$

where Δ_S corresponds to the model skills that are transformed so as to express dissimilarity. The main difference to conventional unfolding is thus the inclusion of the within-set proximities.

The iterative procedure employed by SMACOF is described in Algorithm 3. It is guaranteed that Stress is non-increasing and converges to a minimum, which, however, could be local (De Leeuw and Heiser 1977; Groenen and Heiser 1996). The key point in this approach is the update step for the current configuration, the so-called *Guttman transform*. This step determines the descent direction of the majorized Stress function (39). More details on the derivation can be found in (Borg and Groenen 2005; De Leeuw and Mair 2009; Groenen and van de Velden 2016). Our implementation of dyadic unfolding is based on a Matlab port of smacofSym from the 'smacof' R package (De Leeuw and Mair 2009).

Algorithm 3 SMACOF

Require: Dissimilarities δ, initial point configuration $\boldsymbol{X}_0$
$s_0 \leftarrow \sigma_r(\boldsymbol{X}_0), k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  Update $\boldsymbol{X}_k$ via the **Guttman transform**
  $s_k \leftarrow \sigma_r(\boldsymbol{X}_k)$
until $(s_{k-1} - s_k < \epsilon$ **or** $k = \text{maxiter})$
return $\boldsymbol{X}_k$

7.3 Related visualization approaches for preference ranking data

Besides multidimensional unfolding, other approaches for visualizing ranking data have been developed in the past (Alvo and Yu 2014). For instance, the permutation polytope, its combination with histograms and its projections belong to the classical approaches for visualizing ranking data (Marden 1995). The main disadvantage of the classical approaches is the difficulty of their interpretability with a growing number of ranking items.

There are different visualization techniques based on the vector model of unfolding introduced by Tucker (1960). The main idea is that subjects are represented as so-called preference vectors while items are identified as points. The closer the projection of an item is to a subject vector, the more preferred it is. Techniques that are based on the vector model include MDPref (Carroll and Chang 1970), CATPCA (Meulman et al. 2004), and VIPSCAL (Van Deun et al. 2005). A disadvantage is that the visualization gets confusing as more and more preference vectors enter the scene.

More recently, Kidwell et al. (2008) proposed a visualization technique using MDS in conjunction with Kendall distance on complete and partial rankings. The visualizations are capable of highlighting clusterings by utilizing heat map density plots. An advantage of the newly proposed dyadic unfolding visualization over this and all other techniques is its capability of visualizing probabilistic information.

8 Experiments on dyad ranking

The following experiments are intended to evaluate the performance of our dyad ranking methods BilinPL and PLNet, as well as the usefulness of the dyad ranking setting itself. Thus, we are interested in conditions under which learning can benefit from taking additional label descriptions into account. Our focus is on the specific case of contextual dyad ranking and its comparison to standard label ranking.

In addition to BilinPL and PLNet, we included LinPL (as implemented by Cheng et al. (2010a)) as well as the following state-of-the-art label ranking methods as baselines: Ranking by Pairwise Comparison (RPC, Hüllermeier et al. 2008) and Constrained Classification (CC, Har-Peled et al. 2002a, b), both with logistic regression as base learner.[7] For comparing with conditional ranking, we used QueryRankRLS as implemented in the software package RLScore (Pahikkala and Airola 2016). BilinPL, PLNet and other dyad ranking related methods are provided in the software package DyraLib.[8] BilinPL and PLNet are realized

[7] CC was used in its online variant as described in (Hüllermeier et al. 2008).

[8] https://github.com/disc5/DyraLib.

in Matlab, and as a proof of concept, the latter is also implemented in Python based on the TensorFlow deep learning framework (Abadi et al. 2015).

Predictive performance is measured in terms of Kendall's tau coefficient (Kendall 1938), a rank correlation measure commonly used for this purpose in the label ranking literature (Vembu and Gärtner 2010; Zhou et al. 2014). It is defined as

$$\tau = \frac{C(\pi, \hat{\pi}) - D(\pi, \hat{\pi})}{K(K-1)/2}, \tag{41}$$

with C and D the number of concordant (put in the same order) and discordant (put in the reverse order) label pairs, respectively, and K the length of the rankings π and $\hat{\pi}$ (number of labels). Kendall's tau takes values in $[-1, +1]$, with $\tau = +1$ for the perfect prediction $\hat{\pi} = \pi$ and $\tau = -1$ if $\hat{\pi}$ is the exact reversal of π.

8.1 Learning curves on synthetic data

Ideal synthetic ranking data is created by sampling from the Plackett–Luce distribution according to the BilinPL model specification under the setting (5) of contextual dyad ranking. A realistic scenario is simulated in which labels can be missing, i.e., observed rankings are incomplete (Cheng et al. 2013). To this end, a biased coin is flipped for every label, and it is decided with probability $p \in [0, 1]$ to keep or to delete it. We choose a missing rate of $p = 0.3$, which means that on average 70% of all labels of the training set are kept while the remaining labels are removed. Feature vectors of dimensionality $c = 4$ for labels and dimensionality $r = 3$ for instances were generated by sampling the elements from a standard normal distribution (except for one instance feature, which is a constant). The weight components were sampled randomly from a normal distribution with mean 1 and standard deviation 9. The predictive performance is then determined on a sufficiently large number of (complete) test examples and averaged over 10 repetitions. Note that CC, LinPL, and RPC are standard label ranking methods that do not exploit any attribute information on labels.

The learning curves produced are shown in Fig. 4 for different numbers of labels. Overall, all ranking methods are able to learn and predict correctly if enough training data are available. In the limit, they all reach the performance of the "ground truth": given complete knowledge about the true PL model, the optimal (Bayes) prediction is the mode of that distribution (note that the average performance of that predictor is still not perfect, since sampling from the distribution will not always yield the mode). As expected, BilinPL benefits from the additional label description compared to the other label ranking approaches over a wide range of different training set sizes and numbers of labels. In comparison with the other approaches, the learning curves of PLNet and RankRLS are less steep and require careful tuning of their regularization parameters.

8.2 Label ranking on semi-synthetic UCI data

Label ranking algorithms are commonly evaluated using a suite of benchmark data sets introduced by Cheng et al. (2009). The data is semi-synthetic in the sense of being based on real multi-class and regression data sets taken from the UCI Machine Learning Repository (Lichman 2013), which are modified so as to fit the setting of label ranking; see Table 1 for the main properties of that data. For the multi-class data sets, rankings were generated by training a naive Bayes classifier on the complete data set and ordering the class labels according to the predicted class probabilities for each example. For the regression data sets, a subset of

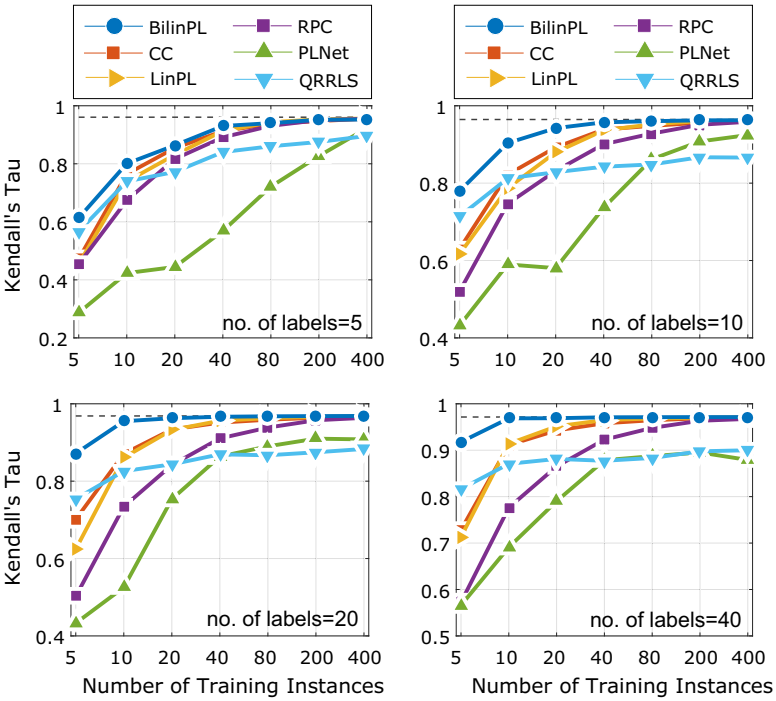

Fig. 4 Learning curves (generalization performance as a function of the number of training examples) of the ranking methods for different numbers of labels

Table 1 Semi-synthetic label ranking data sets and their properties

Classification				Regression			
Data set	# Inst.(N)	# Attr. (d)	# Labels (M)	Data set	# Inst. (N)	# Attr. (d)	# Labels (M)
Authorship	841	70	4	Bodyfat	252	7	7
Glass	214	9	6	Calhousing	20,640	4	4
Iris	150	4	3	Cpu-small	8192	6	5
Pendigits	10,992	16	10	Elevators	16,599	9	9
Segment	2310	18	7	Fried	40,769	9	5
Vehicle	846	18	4	Housing	506	6	6
Vowel	528	10	11	Stock	950	5	5
Wine	178	13	3	Wisconsin	194	16	16

instance attributes were removed from the data and interpreted as labels. Rankings were then obtained by standardizing the attributes and then ordering them by size. This approach is justified by assuming that the original attributes are correlated and the remaining features contain information about the values and hence the ranking of the removed attributes.

Table 2 Results on the UCI label ranking data sets (average Kendall $\tau \pm$ standard deviation)

Data set	BilinPL	CC	PLNet	QRRLS	RPC-LR
Authorship	**0.931 ± 0.013**	0.916 ± 0.015	0.908 ± 0.025	0.432 ± 0.043	0.917 ± 0.020
Bodyfat	0.268 ± 0.059	0.245 ± 0.052	0.251 ± 0.040	0.284 ± 0.057	**0.285 ± 0.061**
Calhousing	0.220 ± 0.011	0.254 ± 0.009	**0.272 ± 0.014**	0.215 ± 0.011	0.243 ± 0.010
Cpu-small	0.445 ± 0.016	0.468 ± 0.017	**0.500 ± 0.019**	0.376 ± 0.012	0.449 ± 0.016
Elevators	0.730 ± 0.007	0.770 ± 0.009	**0.788 ± 0.009**	0.570 ± 0.007	0.749 ± 0.008
Fried	0.999 ± 0.000	0.999 ± 0.000	0.951 ± 0.010	0.996 ± 0.001	**1.000 ± 0.000**
Glass	0.835 ± 0.072	0.830 ± 0.079	0.846 ± 0.080	0.818 ± 0.075	**0.889 ± 0.057**
Housing	0.655 ± 0.040	0.639 ± 0.044	**0.703 ± 0.033**	0.579 ± 0.038	0.672 ± 0.041
Iris	0.813 ± 0.112	0.800 ± 0.109	**0.960 ± 0.049**	0.800 ± 0.064	0.911 ± 0.047
Pendigits	0.892 ± 0.003	0.896 ± 0.002	0.905 ± 0.005	0.561 ± 0.003	**0.932 ± 0.002**
Segment	0.903 ± 0.008	0.910 ± 0.008	**0.939 ± 0.008**	0.720 ± 0.011	0.929 ± 0.009
Stock	0.704 ± 0.016	0.714 ± 0.016	**0.882 ± 0.020**	0.663 ± 0.016	0.774 ± 0.024
Vehicle	0.855 ± 0.020	0.850 ± 0.025	**0.872 ± 0.025**	0.776 ± 0.031	0.855 ± 0.015
Vowel	0.581 ± 0.026	0.577 ± 0.046	**0.805 ± 0.016**	0.574 ± 0.026	0.644 ± 0.021
Wine	0.929 ± 0.052	0.914 ± 0.069	**0.942 ± 0.034**	0.923 ± 0.065	0.925 ± 0.054
Wisconsin	0.629 ± 0.028	0.612 ± 0.030	0.514 ± 0.028	0.630 ± 0.031	**0.632 ± 0.027**
Average rank	3.063	3.438	2.000	4.500	2.000

The winning method per dataset is indicated in bold face

8.2.1 Experimental results

We compare the performance of BilinPL and PLNet to other state-of-the-art label ranking methods using 10-fold cross-validation. For PLNet, we use three layers with 10 neurons for the hidden layer. Labels were encoded for BilinPL and PLNet in terms of 1-of-K dummy vectors without utilizing further label descriptions. For BilinPL, an additional bias term (constant 1) is added to the representation of instances.

The results provided in Table 2 suggest that PLNet is highly competitive, even compared with RPC, which is known for its strong performance. Most probably, this is due to its ability to model latent utilities in a flexible, nonlinear way. The price to pay is an increased danger of overfitting in cases where this flexibility is not needed. This can indeed be seen on data sets with many attributes but a small number of instances. Linear models are advantageous in comparison to PLNet if their inductive bias is correct, which is the case, for example, for the fried data.

8.2.2 Unfolding of label ranking predictions

We use the PLNet model on the example of the vowel data set to construct an unfolding. Being capable of modeling non-linear relationships, PLNet was able to outperform all other methods specifically on this data set. The vowel data consists of 528 instances and 11 labels. The data is split in the ratio 90/10 for training and test, and the transformation t_2 was applied. The new ideal point configuration was obtained after 123 iterations with a stress value of 0.1630 under the setting $\alpha = \beta = 0$ and $\gamma = 0.8$ in (40).

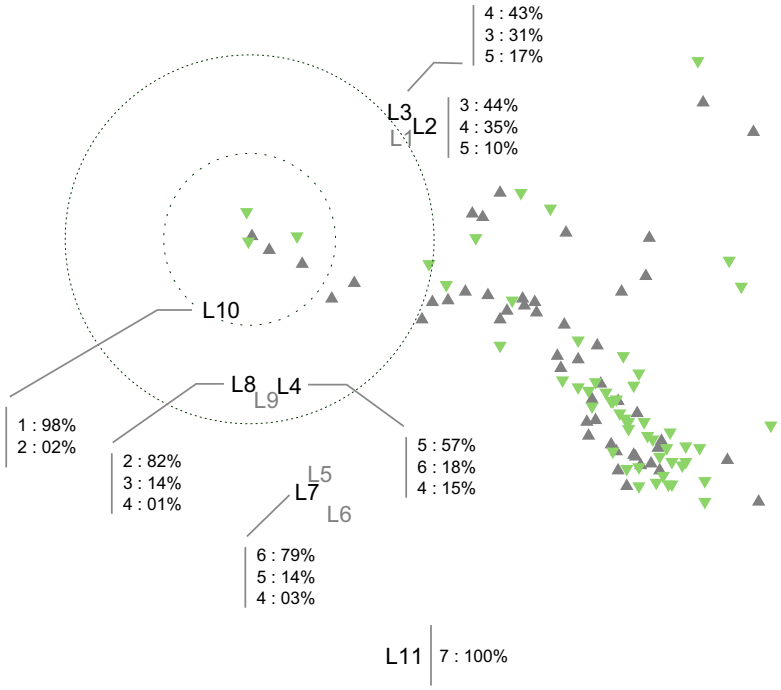

Fig. 5 Unfolding representation of modeled label rankings of the Vowel data set with PLNet. Triangles denote instances (upright/gray: training, upside-down/green: testing) and class labels are indicated with L[1–11]. The circles indicate isopreference contours for a test instance in their center. The class labels (vowels) are arranged in such a way that those which are closer to the test instance are better suited than those that are farer away. The uncertainty that comes along with such an arrangement is indicated by probabilistic information of the form $x : y$, where x denotes a rank position and y is the corresponding probability. These numbers are calculated for that particular test instance relating to the labels and are given in Table 3

In Fig. 5, the isopreference contours are drawn for a particular test instance denoted as a triangle. The mode predicted by our model for this instance is the ranking $L10 \succ L8 \succ L9 \succ L1 \succ L2 \succ \cdots$, which is reflected by the unfolding reasonably well. In the figure, the unfolding is supplemented by a list of top-3 positions (with corresponding probabilities) for each label according to (38). The complete marginal distributions are given in Table 3, with the values shown in the visualization highlighted in bold font.

8.3 Multi-label ranking of musical emotions

In this experiment, PLNet is used to rank labels that are specified in a multi-label classification context. The Emotions dataset is about songs that were annotated by experts using multiple emotional labels based on the Tellegen-Watson-Clark model (Trohidis et al. 2008). In total, 593 songs from a variety of genres (Classical, Reggae, Rock, Pop, Hip-Hop, Techno, Jazz) were used from which 72 rythmic as well as timbre features were extracted and used as a feature representation.

To apply dyad ranking on this kind of multi-label data, a preference $(\boldsymbol{x}, y) \succ (\boldsymbol{x}, y')$ is constructed for each song $\boldsymbol{x}$ and each pair of emotions y, y' such that y is associated with (or relevant for) $\boldsymbol{x}$ but y' is not. Once being trained on this data, a dyad ranker will be able to predict a ranking of emotions, contextualized by a song. Note, however, that such as ranking

Table 3 Marginal distributions on a subset of labels for the highlighted test instance in Fig. 5

	Ranks						
	1	2	3	4	5	6	7
Labels							
2	0.001 [6]	0.086 [4]	**0.445 [1]**	**0.356 [2]**	**0.104 [3]**	0.008 [5]	0.000 [7]
3	0.001 [6]	0.059 [4]	**0.310 [2]**	**0.438 [1]**	**0.174 [3]**	0.018 [5]	0.000 [7]
4	0.000 [6]	0.015 [5]	0.078 [4]	**0.151 [3]**	**0.574 [1]**	**0.182 [2]**	0.000 [7]
7	0.000 [6]	0.004 [5]	0.019 [4]	**0.038 [3]**	**0.148 [2]**	**0.791 [1]**	0.000 [7]
8	0.013 [4]	**0.821 [1]**	**0.149 [2]**	**0.016 [3]**	0.001 [5]	0.000 [6]	0.000 [7]
10	**0.985 [1]**	**0.015 [2]**	**0.000 [3]**	0.000 [4]	0.000 [5]	0.000 [6]	0.000 [7]
11	0.000 [7]	0.000 [6]	0.000 [5]	0.000 [4]	**0.000 [3]**	**0.000 [2]**	**1.000 [1]**

The ranking of rank positions is indicated via square brackets besides the probabilities

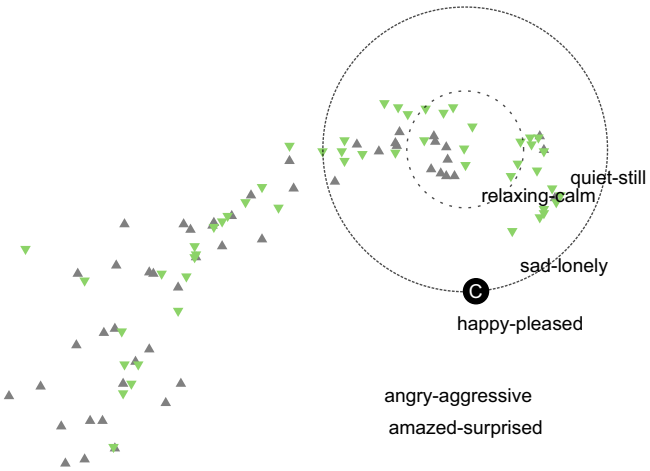

Fig. 6 Unfolding representation of modeled multi-label rankings of the Emotions data set with PLNet. Triangles denote instances (upright/gray: training, upside-down/green: testing) and six class labels. For a test instance which is located in the center of the two dotted circles, a special calibration label (C) divides the set of labels in two groups. Labels closer to the test instance are more appropriate than those that are more distant (Color figure online)

only provides a relative order, but no (absolute) separation of relevant and non-relevant labels. This problem has been addressed by a technique called *calibrated* label ranking (Brinker et al. 2006; Fürnkranz et al. 2008): An additional calibration label is introduced, which models exactly this separation. Correspondingly, preferences of all relevant labels over the calibration label and of the calibration label over all non-relevant labels are added to the training data.

PLNet is trained on 391 examples. The features were normalized by scaling them to [0, 1], and the six labels including the calibration label were encoded using 1-of-k encoding. The network consists of one hidden layer with 10 neurons and is trained using the procedure described above. The trained PLNet is then used to make predictions on 202 test examples. Training and test skills are used for dyadic unfolding with $\alpha = 0$, $\beta = 0$, $\gamma = 1$. After 90 iterations of SMACOF, a final stress value of 0.0560 and a point configuration was obtained, which is shown in Fig. 6.

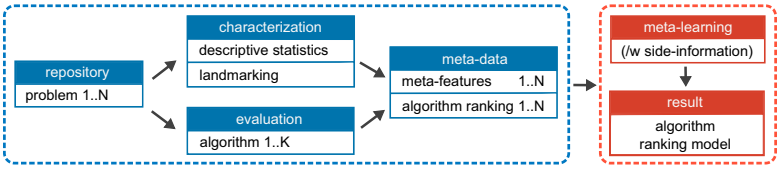

Fig. 7 The components of the "meta-learning for algorithm recommendation" framework shown above are based on (Brazdil et al. 2008). The left box shows the meta-data acquisition process which consists of the problem (or data set) description and the evaluation of the algorithms on the problems (or data sets). The box on the right side, the meta-level learning part, shows the meta-learning process and its outcome. In this case study, the meta-learner must be able to deal with qualitative data in the form of rankings and is furthermore allowed to use additional knowledge (side-information) about the algorithms if available

As can be seen, the unfolding nicely reflects the similarity between emotions. For example, *quiet-still* is located much closer to *relaxing-calm* than ro *happy-pleased*. Likewise, *angry-aggressive* and *amazed-surprised* are close to each other but quite far from the other emotions. Overall, the unfolding suggests a spectrum ranging from *quiet-still* to *amazed-surprised*, and the songs are distributed along that spectrum. The absolute fit seems to be better for the left side of the spectrum, as can be seen from the difference between the songs and the emotions.

8.4 Algorithm configuration

In the following experiment, we apply our dyad ranking methods in the setting of meta-learning for algorithm recommendation as described by Brazdil et al. (2008). In particular, given a problem instance, we aim at predicting rankings of configurations of genetic algorithm (GA). The dyad ranking setting is here explored in full generality, because rankings of different length are considered as well as features about instances (problems) and labels (algorithms).

The meta-learning framework provides several degrees of freedom, including the way in which meta-data is acquired (see Fig. 7). The meta-features as part of the meta-data should be able to relate a problem instance (e.g., a data set) to the relative performance of the candidate algorithms. They are usually made up by a set of numbers acquired by using descriptive statistics. Another possibility consists of probing a few parameter settings of the algorithm under consideration. The performance values of those *landmarkers* can then be used as instance-features for the meta-learner. In addition to the meta-features, the meta-data consists of rankings of the configurations, i.e., a sorting of the variants in decreasing order of performance.

In analogy to the majority classifier typically used as a baseline in multi-class classification, the meta-learning literature suggests a simple approach called the Average Ranks (AR) method (Brazdil et al. 2008). This approach corresponds to what is called the Borda count in the ranking literature and produces a default prediction by sorting the alternatives according to their average position in the observed rankings. The AR of a configuration a_j, $1 \leq j \leq M$, which occurs in N_j of the N rankings, is defined as

$$\overline{R}_j = \frac{\sum_{i=1}^{N_j} R_{i,j}}{N_j},$$

where $R_{i,j}$ is the rank of a_j within the ith ranking (in which it occurs).

In this experiment we focus on the task of *ranking* configurations for genetic algorithms (GAs). These GAs are applied on different instances of the traveling salesman problem (TSP). For the training, the GA performance averages are taken to construct rankings, in which a

single performance value corresponds to the distance of the shortest route found by a GA. The GAs all share the properties of using the same selection criterion, which is "roulette-wheel", the same mutation operator, which is "exchange mutation", and "elitism" of 10 chromosomes (Mitchell 1998).

We tested the performance of three groups of GAs on a set of TSP instances. The groups are determined by their choice of the crossover operator, which are cycle (CX), order (OX), or partially mapped crossover (PMX) (Larrañaga et al. 1999). Problem instances are characterized by the number of cities and the performances of three landmarkers.

In total, 246 problems are considered, with the number of cities ranging between 10 and 255. For each problem, the city locations (x, y) are drawn randomly from the uniform distribution on $[1, 100]^2$. Moreover, 72 different GAs are considered as alternatives with their parameters as optional label descriptions. They share the number of generations, 500, and the population size of 100. The combinations of all the other parameters, namely, crossover type, crossover rate, and mutation rate, are used for characterization:

- Crossover types: {CX, OX, PMX}
- Crossover rates: {0.5, 0.6, 0.7, 0.8, 0.9}
- Mutation rates: {0.08, 0.09, 0.1, 0.11, 0.12}

The three landmarker GAs have a crossover rate of 0.6 and a mutation rate of 0.12, combined with one of the three crossover types, respectively. They are excluded from the set of alternatives to be ranked. The label and dyad rankers are faced with rankings under different conditions (M, N), with N the number of training instances and M the average length of the rankings (M of the 72 alternatives are chosen at random while the others are discarded).

8.4.1 Experimental results

The results in Table 4 are quite consistent with our previous studies. Again, they confirm that additional information about labels can principally be exploited by a learner to achieve better predictive performance. In particular, BilinPL is able to take advantage of this information for small values of M and compares favorably to the other label rankers (and, in addition, has of course the advantage of being able to rank GA variants that have not been used in the training phase). As expected, standard label rankers (in this case, CC) surpass BilinPL only for a sufficiently large amount of training data. PLNet is superior to other approaches in cases of many labels and instances.

8.4.2 Unfolding of algorithm configurations

We used the bilinear PL model to create an unfolding solution for the meta-learning problem of recommending rankings over genetic algorithm configurations. To this end, we first trained the BilinPL model on 120 examples, each of which provides an incomplete ranking over only 5 of the 72 random configurations. The model was then used to predict the rankings on 126 new problem instances, and to complement the missing ranking information about the 120 training examples. Dyadic unfolding was then performed (using transformation t_2) and the pairwise within-set distances of points in X and Y. The calculation of the ideal point configuration under the setting of $\alpha = 0.1$, $\beta = 0.1$, and $\gamma = 0.8$ required 24 SMACOF iterations with a final stress value of 0.0975.

The resulting unfolding configuration shown in Fig. 8 nicely reveals the degree of suitability of GA configurations for particular TSP problems: GAs of type OX and PMX are more suitable for smaller problems, while GAs of the type CX are better suited for TSPs

Table 4 Average performance in terms of Kendall's tau and standard deviations of different meta-learners and different conditions (average rankings lengths M and the numbers of training instances N)

M	N	AR	BilinPL	CC	LinPL	PLNet	QRRLS	RPC
5	30	.192 ± .063	**.727 ± .014**	.290 ± .063	.317 ± .049	.602 ± .057	.598 ± .041	.158 ± .052
	60	.358 ± .046	**.766 ± .014**	.428 ± .040	.452 ± .041	.651 ± .049	.633 ± .021	.311 ± .038
	90	.404 ± .030	**.770 ± .014**	.573 ± .042	.575 ± .037	.685 ± .063	.634 ± .028	.372 ± .035
	120	.430 ± .029	**.777 ± .009**	.610 ± .031	.619 ± .022	.727 ± .031	.644 ± .025	.387 ± .032
10	30	.423 ± .054	**.775 ± .007**	.539 ± .054	.551 ± .049	.724 ± .035	.634 ± .018	.397 ± .043
	60	.487 ± .017	**.781 ± .004**	.690 ± .021	.696 ± .013	.744 ± .022	.652 ± .022	.493 ± .037
	90	.523 ± .014	**.781 ± .007**	.726 ± .015	.726 ± .012	.774 ± .010	.657 ± .024	.576 ± .018
	120	.522 ± .015	**.783 ± .006**	.750 ± .014	.748 ± .014	.783 ± .017	.661 ± .026	.620 ± .020
20	30	.516 ± .037	**.781 ± .005**	.722 ± .019	.722 ± .015	.747 ± .037	.659 ± .016	.622 ± .018
	60	.549 ± .014	.784 ± .005	.763 ± .013	.758 ± .014	**.787 ± .010**	.659 ± .024	.714 ± .022
	90	.561 ± .014	.787 ± .006	.779 ± .010	.774 ± .013	**.793 ± .013**	.656 ± .033	.751 ± .021
	120	.571 ± .022	.787 ± .008	.786 ± .010	.782 ± .010	**.794 ± .012**	.659 ± .036	.772 ± .014
30	30	.554 ± .028	**.782 ± .005**	.753 ± .013	.746 ± .018	.772 ± .019	.655 ± .018	.717 ± .019
	60	.567 ± .008	.785 ± .003	.782 ± .007	.775 ± .009	**.791 ± .008**	.661 ± .020	.767 ± .011
	90	.578 ± .008	.787 ± .004	.791 ± .005	.786 ± .005	**.798 ± .003**	.663 ± .015	.781 ± .006
	120	.580 ± .011	.786 ± .006	.794 ± .005	.789 ± .007	**.799 ± .007**	.666 ± .024	.787 ± .005

The best performance per condition is indicated in bold

with a larger number of cities. Moreover, the types of GA are reflected well by the different clusters. Each cluster is again (sub-)clustered according to mutation rates as indicated by different hues of colors. Isocontour circles are drawn exemplarily around a particular instance (here training problem 6 with 166 cities) to support the inspection of the GA ranking.

8.5 Similarity learning on tagged images

In this case study, we apply dyad ranking for *similarity learning* on image data, and make use of our visualization technique for unfolding image similarities. Observations are in the form of pairwise preferences over dyads (i.e., rankings of length 2), and all dyad members stem from a common domain. The images under consideration are taken from the Caltech256 data set, and each of them belongs to one of several categories (Griffin et al. 2007). The images mainly contain a single object, so that images can be identified with classes quite unambiguously.

8.5.1 Learning image similarity using dyad ranking

A collection of images, in which each image is tagged with a class label, can be used to infer a notion of preference. For learning a measure of similarity, we make the reasonable assumption that a pair of images sharing the same label are mutually more similar to each other than a pair of images with different labels.

Given a finite set of class labels, there are multiple ways to construct dyad rankings based on this idea, as depicted in Fig. 9. In panels (a)–(c), there are dyad rankings which have in common that the first ranked dyad contains instances that are similar and the second ranked dyad contains instances that are dissimilar. These rankings are thus of the form

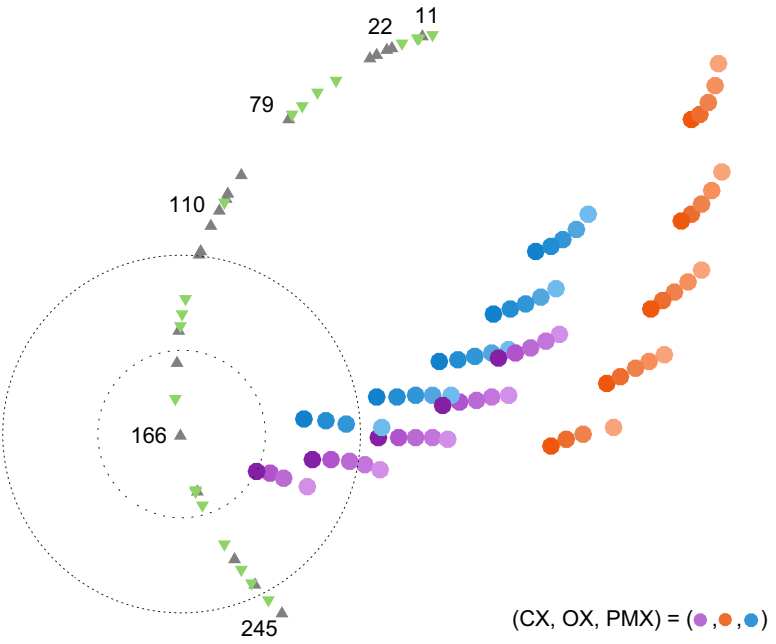

Fig. 8 Unfolding of GA configuration preferences modeled with BilinPL. Triangles reflect training (upright/gray) and testing (upside-down/green) TSP instances, and circles refer to GA configurations. A random selection of instances are annotated with their associated numbers of cities. Instances are distributed across a curved formation with fewer cities at one end and many at the other end. The circles around a training instance with 166 cities represent isopreference contours, which help identifying suitable solutions (Color figure online)

$(\boldsymbol{x}_1, \boldsymbol{x}_2) \succ (\boldsymbol{x}_3, \boldsymbol{x}_4)$. Panel (b) is a special case of (a) in which one of the second ranked dyad instances belongs to the same class as those from the first ranked dyad. Panel (c) is again a special case of (b) in which one of the instances of the second ranked dyad coincides with one of the instances of the first ranked dyad. This can be translated into a contextualized preference statement "instance $\boldsymbol{x}_1$ is more similar to instance $\boldsymbol{x}_2$ than to $\boldsymbol{x}_3$" (Cheng and Hüllermeier 2008).

Since the extraction of all possible dyad rankings from the multi-class data would lead to a prohibitively large data set for conventional maximum likelihood estimation procedures (such as L-BFGS), the model is learned in an *online* fashion by sampling dyad rankings on the fly (cf. Sect. 5.3.3). This online approach is called SiDRa, which stands for "Similarity Learning using Dyad Ranking".

An important feature of the learning algorithm is the possibility of combining it with a selective sampling strategy (cf. Algorithm 4), which leverages the probabilistic information provided by the (bilinear) PL model. Roughly speaking, by selecting dyad pairs for which the (predicted) preference is highly uncertain, this strategy implements a simple form of uncertainty sampling. Compared to simple random sampling, it results in a speed up of the learning process, because an equivalent model can be constructed with fewer parameter updates.

The implementation is also available in DyraLib and is based on Matlab with parts written in C++. This approach is also encouraged by existing metric learning approaches, which

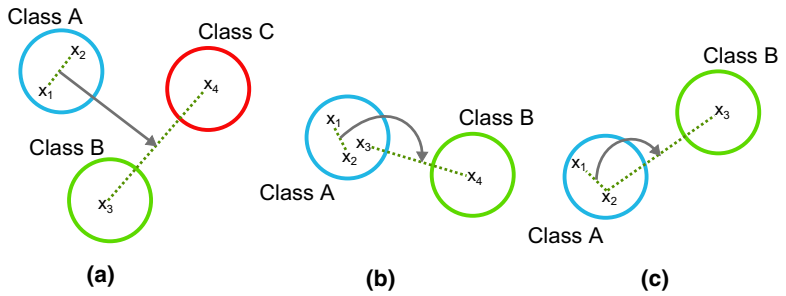

Fig. 9 Elicitation of dyad rankings from multi-class data. **a–c** Different ways to find dyad rankings of length 2 in which the dyad at the first rank position contains instances which are more similar to each other compared to the instances contained in the dyad put at the second rank position

Algorithm 4 Selective Sampling with Bilinear PL

procedure SELECTIVESAMPLING($\mathcal{S}$, $\boldsymbol{w}$)
  $[\boldsymbol{x}', \boldsymbol{y}'] \leftarrow$ **sampleDyad**($\mathcal{S}$)
  repeat
    $[\boldsymbol{x}'', \boldsymbol{y}''] \leftarrow$ **sampleDyad**($\mathcal{S}$) ▷ See constraints (a)–(c) in Fig. 9.
    $\boldsymbol{x} \leftarrow [\boldsymbol{x}' \otimes \boldsymbol{y}' - \boldsymbol{x}'' \otimes \boldsymbol{y}'']$
  until (**uncertaintyHigh**($\boldsymbol{w}$, $\boldsymbol{x}$)))
  return $(\boldsymbol{x}', \boldsymbol{y}') \succ (\boldsymbol{x}'', \boldsymbol{y}'')$
end procedure

typically use training examples of type (c) in Fig. 9 (Bellet et al. 2013). To this end, of course, a suitable feature representation for the images is needed.

8.5.2 Construction of image features

The Caltech256 data set has already been used for image similarity learning before (Chechik et al. 2009, 2010). The authors of these papers used a feature engineering approach, in which images are represented as sparse bag-of-visual words vectors. Taking advantage of recent progress in deep learning, we utilize deep convolutional neural networks (CNN) for generating feature representations. More concretely, we used a pre-trained CNN model called AlexNet, which has been created on the basis of the ImageNet data set (Krizhevsky et al. 2012; Jia et al. 2014). For each image, 4096 dimensional sparse feature vectors were obtained from the outputs of the 6th fully connected (6fc) layer of the convolutional neural network.

8.5.3 Unfolding of image similarity

SiDRa was run with a learning rate $\eta = 0.1$ and a regularization parameter $\lambda = 0.0001$. The dyad rankings were obtained during the learning process by sampling images involving all possible cases (a)–(c) as outlined in Fig. 9. The final BilinPL model comprised a weight matrix of 16 million elements ($= 4096^2$) and took 30 K iterations. The model was then used to produce a matrix of skill values for images in the test set. The rank-transform (t_3) of the skills was used for dyadic unfolding, which produced a configuration of points within 339 iterations.

The resulting unfolding solution is shown in Fig. 10. As can be seen, images sharing the same tags tend to be members of the same clusters. One example of an interpretation of

Fig. 10 Multidimensional unfolding solution of a dyad ranking model. Images that share the same tag are grouped together in a 2 dimensional plane

the resulting configuration is as follows: Things with corners are located at the right side, whereas roundish things are found on the left side.

The Pearson product moment correlation coefficient on pairwise log scores $v(\boldsymbol{x}, \boldsymbol{x}')$, i.e., bilinear similarities, and pairwise column configuration point distances amounts to $r = -0.791$. This means, the lower the distance between two images in the unfolding solution, the higher their dyadic preference in terms of similarity.

9 Conclusion

In this paper, we proposed dyad ranking as an extension of the label ranking problem, a specific problem in the realm of preference learning, in which preferences on a finite set of choice alternatives are represented in the form of a contextualized ranking. While the context is described in terms of a feature vector, the alternatives are merely identified by their label.

In practice, however, information about properties of the alternatives is often available, too, and such information could obviously be useful from a learning point of view. In dyad ranking, not only the context but also the alternatives are therefore characterized in terms of properties. The key notion of a *dyad* refers to a combination of a context and a choice alternative.

We proposed a method for dyad ranking that is an extension of an existing probabilistic label ranking method based on the Plackett–Luce model. This model is combined with a suitable feature representation of dyads. Concretely, we developed two instantiations of this approach. The first, BilinPL, is a method based on the representation of dyads in terms of a Kronecker product. The second, PLNet, is based on neural networks and allows for learning a (highly nonlinear) joint feature representation of dyads. The usefulness of the additional information provided by the feature description of alternatives was shown in several experiments and case studies on algorithm configuration and similarity learning.

Last but not least, we proposed a method for the visualization of dyad rankings, which is based on the technique of multidimensional unfolding. We consider this as an interesting contribution, especially because visualization has hardly been studied in the field of preference learning so far. Again, the usefulness of our approach was shown in several case studies.

There are several directions for future work. We are specifically interested in developing a "deep" version of PLNet. So far, PLNet is based on standard multilayer networks. However, in many applications, it could be advantageous to replace these networks by deep architectures. In our case study on image data, deep features were first extracted manually from a convolutional neural net, and then used for dyad ranking in a second step. A much more elegant approach, of course, would be to combine these steps in a single method. PLNet with a deep neural net is an obvious candidate for such a method.

A Proof of Proposition 1

Recall that the basic PL model is invariant against multiplication with a positive constant, and that this is the only invariance of the model. Since the bilinear PL model defined by (14) is log-linear in $\mathbf{W}$, invariance on the level of this parameter can only be additive. Now, suppose there are two parameters $\mathbf{W} \neq \mathbf{W}^*$ that both induce the same distribution on the set of all potential dyad subsets, which means that

$$\boldsymbol{x}^\top \mathbf{W} \boldsymbol{y} = \boldsymbol{x}^\top \mathbf{W}^* \boldsymbol{y} + \gamma \tag{42}$$

for all dyads $(\boldsymbol{x}, \boldsymbol{y})$, where γ is a constant that may depend on the parameters $\mathbf{W}$ and $\mathbf{W}^*$ but *not* on the dyads $(\boldsymbol{x}, \boldsymbol{y})$. More specifically, for the case of contextual dyad ranking, γ is also allowed to depend on $\boldsymbol{x}$, but again, must not depend on $\boldsymbol{y}$. Under our assumptions, however, this independence cannot hold. In fact, denoting the elements of $\mathbf{W}$ and $\mathbf{W}^*$ by $w_{i,j}$ and $w^*_{i,j}$, respectively, (42) means that

$$\sum_{i=1}^{r} \sum_{j=1}^{c} (w_{i,j} - w^*_{i,j}) x_i y_j = \sum_{i=1}^{r} \sum_{j=1}^{c} \Delta w_{i,j} x_i y_j = \gamma.$$

Then, exploiting the fact that not all $\Delta w_{i,j}$ can vanish at the same time, it is not difficult to show that a variation of some values y_j, which will also have an influence on the difference γ, is always possible.

B Convexity of the NLL (18)

To show the convexity of the negative log-likelihood function (18), recall that

$$\ell_n(\boldsymbol{w}; \mathcal{D}) = \sum_{n=1}^{N} - \log P(\rho_n \mid \boldsymbol{w}) \tag{43}$$

with

$$- \log P(\rho \mid \boldsymbol{w}) = \sum_{m=1}^{M-1} \underbrace{- \boldsymbol{w}^\top \boldsymbol{z}^{(m)}}_{\text{term 1}} + \sum_{m=1}^{M-1} \underbrace{\log \left(\sum_{l=m}^{M} \exp(\boldsymbol{w}^\top \boldsymbol{z}^{(l)}) \right)}_{\text{term 2}}, \tag{44}$$

where the index n is dropped in ρ_n and $\boldsymbol{z}^{(n,l)}$ for better readability. Using the fact (e.g. from Luenberger (1973)) that a function that consists of a sum of convex functions is convex again,

it needs thus to be analyzed if each of the summands in (44) is convex to state the convexity for $\ell_n(\mathcal{D}, \boldsymbol{w})$. Term 1 is a linear function and is convex and concave (in a non strictly sense) at the same time. The term 2 refers to the function

$$f(\boldsymbol{w}) := \log\left(\sum_l \exp(\boldsymbol{w}^\top z^{(l)})\right), \tag{45}$$

for which we analyze its convexity by restricting it to an arbitrary line (see Boyd and Vandenberghe 2004, Chapter 3). To this end we specify the function

$$\begin{aligned} g(t) := f(\boldsymbol{w} + t\boldsymbol{v}) &= \log\left(\sum_l \exp((\boldsymbol{w} + t\boldsymbol{v})^\top z^{(l)})\right) \\ &= \log\left(\sum_l \exp(\boldsymbol{w}^\top z^{(l)} + t\boldsymbol{v}^\top z^{(l)})\right). \end{aligned} \tag{46}$$

A criterion for the convexity of $g(\cdot)$ to be convex is that its second derivative is always positive. The first derivative of (46) is given by

$$g'(t) = h(t)^{-1}\left[\sum_l \boldsymbol{v}^\top z^{(l)} \exp(\boldsymbol{w} z^{(l)} + t\boldsymbol{v}^\top z^{(l)})\right].$$

with

$$h(t) = \sum_l \exp\left(\boldsymbol{w} z^{(l)} + t\boldsymbol{v}^\top z^{(l)}\right).$$

The second derivative can be stated then as follows:

$$\begin{aligned} g''(t) = h(t)^{-2}\Bigg[&\sum_l (\boldsymbol{v}^\top z^{(l)})^2 \exp\left(\boldsymbol{w}^\top z^{(l)} + t\boldsymbol{v}^\top z^{(l)}\right) \cdot \sum_k \exp\left(\boldsymbol{w}^\top z^{(k)} + t\boldsymbol{v}^\top z^{(k)}\right) \\ &- \sum_l \boldsymbol{v}^\top z^{(l)} \exp(\boldsymbol{w}^\top z^{(l)} + t\boldsymbol{v}^\top z^{(l)}) \cdot \boldsymbol{v}^\top z^{(k)} \sum_k \exp\left(\boldsymbol{w}^\top z^{(k)} + t\boldsymbol{v}^\top z^{(k)}\right)\Bigg] \\ = h(t)^{-2}\Bigg[&\sum_l \sum_k (\boldsymbol{v}^\top z^{(l)})^2 \cdot \xi(\boldsymbol{w}, \boldsymbol{v}, t) - \sum_l \sum_k \boldsymbol{v}^\top z^{(l)} \cdot \boldsymbol{v}^\top z^{(k)} \cdot \xi(\boldsymbol{w}, \boldsymbol{v}, t)\Bigg], \end{aligned}$$

with $\xi(\boldsymbol{w}, \boldsymbol{v}, t) = \exp\left(\boldsymbol{w}^\top (z^{(l)} + z^{(k)}) + t\boldsymbol{v}^\top (z^{(l)} + z^{(k)})\right)$. For all choices of $\boldsymbol{x}$, $\boldsymbol{v}$ and t we have:

$$\begin{aligned} g''(t) &= h(t)^{-2}\left[\sum_l \sum_k \left[\left(\boldsymbol{v}^\top z^{(l)}\right)^2 - \boldsymbol{v}^\top z^{(l)} \cdot \boldsymbol{v}^\top z^{(k)}\right] \cdot \xi(\boldsymbol{w}, \boldsymbol{v}, t)\right] \\ &= h(t)^{-2}\left[\sum_l \sum_k \left[\frac{\left(\boldsymbol{v}^\top z^{(l)}\right)^2}{2} - \boldsymbol{v}^\top z^{(l)} \cdot \boldsymbol{v}^\top z^{(k)} + \frac{\left(\boldsymbol{v}^\top z^{(k)}\right)^2}{2}\right] \cdot \xi(\boldsymbol{w}, \boldsymbol{v}, t)\right] \\ &= h(t)^{-2}\left[\sum_l \sum_k \left[\frac{\left(\boldsymbol{v}^\top z^{(l)} - \boldsymbol{v}^\top z^{(k)}\right)^2}{2}\right] \cdot \xi(\boldsymbol{w}, \boldsymbol{v}, t)\right] \geq 0 \end{aligned}$$

□

The last argument shows that all possible second derivatives are positive because each involved factor is equal or larger than zero.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. *CoRR*. arXiv:1603.04467.

Alvo, M., & Yu, P. L. (2014). *Statistical methods for ranking data*. New York: Springer.

Bakir, G., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., & Vishwanathan, S. V. N. (Eds.). (2007). *Predicting structured data*. Cambridge: MIT Press.

Basilico, J., & Hofmann, T. (2004). Unifying collaborative and content-based filtering. In *Proceedings ICML, 21st international conference on machine learning*. ACM, New York, USA.

Bellet, A., Habrard, A., & Sebban, M. (2013). A survey on metric learning for feature vectors and structured data (p. 57). arXiv:1306.6709.

Borg, I. (1981). *Anwendungsorientierte Multidimensionale Skalierung*. New York: Springer.

Borg, I., & Groenen, P. (2005). *Modern multidimensional scaling: Theory and applications*. New York: Springer.

Borg, I., Groenen, P. J., & Mair, P. (2012). *Applied multidimensional scaling*. New York: Springer.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of the COMPSTAT'2010, 19th international conference on computational statistics* (pp. 177–187). Springer, Paris, France.

Bottou, L. (1998). *Online algorithms and stochastic approximations. Online learning and neural networks*. Cambridge: Cambridge University Press.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Reading, MA: Cambridge University Press.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2008). *Metalearning: Applications to data mining* (1st ed.). New York: Springer.

Brinker, K., Fürnkranz, J., & Hüllermeier, E. (2006). A unified model for multilabel classification and ranking. In *Proceedings of the ECAI2006: 17th European conference on artificial intelligence* (pp. 489–493), Riva Del Garda, Italy.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., et al. (2005). Learning to rank using gradient descent. In *Proceedings ICML, 22nd international conference on machine learning* (pp. 89–96). ACM, New York, USA.

Busing, F. (2010). Advances in multidimensional unfolding. Ph.D. thesis, University of Leiden.

Cao, Z., Qin, T., Liu, T. Y., Tsai, M. F., & Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. In *Proceedings ICML, 24th international conference on machine learning* (pp. 129–136). ACM, New York, USA.

Caron, F., & Doucet, A. (2012). Efficient bayesian inference for generalized Bradley–Terry models. *Journal of Computational and Graphical Statistics*, *21*(1), 174–196.

Carroll, J. D., & Chang, J. J. (1970). Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart–Young" decomposition. *Psychometrika*, *35*(3), 283–319.

Chechik, G., Sharma, V., Shalit, U., & Bengio, S. (2009). An online algorithm for large scale image similarity learning. *Advances in Neural Information Processing Systems*, *21*, 1–9.

Chechik, G., Sharma, V., Shalit, U., & Bengio, S. (2010). Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, *11*, 1–29.

Cheng, W., & Hüllermeier, E. (2008). Learning similarity functions from qualitative feedback. In *Proceedings of the ECCBR—2008, 9th European conference on case-based reasoning* (pp. 120–134). Springer, Trier, Germany, no. 5239 in LNAI.

Cheng, W., Henzgen, S., & Hüllermeier, E. (2013). Labelwise versus pairwise decomposition in label ranking. In *Proceedings of Lernen Wissen Adaptivität 2013 (LWA 2013)* (pp. 140–147). Otto Friedrich Universität Bamberg, Germany.

Cheng, W., Hühn, J., & Hüllermeier, E. (2009). Decision tree and instance-based learning for label ranking. In *Proceedings ICML, 26th international conference on machine learning* (pp. 161–168). Omnipress, Montreal, Canada.

Cheng, W., Hüllermeier, E., Waegeman, W., & Welker, V. (2012). Label ranking with partial abstention based on thresholded probabilistic models. In *Proceedings NIPS—2012, 26th annual conference on neural information processing systems*, Lake Tahoe, Nevada, US.

Cheng, W., Rademaker, M., De Beats, B., & Hüllermeier, E. (2010b). Predicting partial orders: Ranking with abstention. In *Proceedings ECML/PKDD—2010, European conference on machine learning and principles and practice of knowledge discovery in databases*, Barcelona, Spain.

Cheng, W., Dembczyński, K., & Hüllermeier, E. (2010a). Label ranking methods based on the Plackett–Luce model. In J. Fürnkranz & T. Joachims (Eds.), *Proceedings ICML, 27th international conference on machine learning* (pp. 215–222). Haifa: Omnipress.

Cohen, W., Schapire, R., & Singer, Y. (1999). Learning to order things. *Journal of Artificial Intelligence Research*, *10*(1), 243–270.
Coombs, C. H. (1950). Psychological scaling without a unit of measurement. *Psychological Review*, *57*(3), 145–158.
David, H. A. (1969). *The method of paired comparisons*. London: Griffin.
De Leeuw, J., & Mair, P. (2009). Multidimensional scaling using majorization: SMACOF in R. *Journal of Statistical Software*, *31*(1), 1–30.
De Leeuw, J. (1977). Applications of convex analysis to multidimensional scaling. In J. R. Barra, F. Brodeau, G. Romier, & B. Van Cutsem (Eds.), *Recent developments in statistics* (pp. 133–146). North Holland.
De Leeuw, J., & Heiser, W. J. (1977). Convergence of correction matrix algorithms for multidimensional scaling. In J. C. Lingoes (Ed.), *Geometric representations of relational data* (pp. 735–752). Ann Arbor, MI: Mathesis Press.
Dekel, O., Singer, Y., & Manning, C. D. (2004). Log-linear models for label ranking. In S. Thrun, L. K. Saul, & B. Schölkopf (Eds.), *Advances in neural information processing systems* (Vol. 16, pp. 497–504). Cambridge: MIT Press.
Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society Series B (methodological)*, *39*(1), 1–38.
Fürnkranz, J., & Hüllermeier, E. (2010). Preference learning: An introduction. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference learning* (pp. 1–17). New York: Springer.
Fürnkranz, J., Hüllermeier, E., Mencía, E., & Brinker, K. (2008). Multilabel classification via calibrated label ranking. *Machine Learning*, *73*(2), 133–153.
Griffin, G., Holub, A., & Perona, P. (2007). Caltech-256 object category dataset. *Caltech Mimeo*, *11*, 20.
Groenen, P. J., & Heiser, W. J. (1996). The tunneling method for global optimization in multidimensional scaling. *Psychometrika*, *61*(3), 529–550.
Groenen, P., & van de Velden, M. (2016). Multidimensional scaling by majorization: A review. *Journal of Statistical Software*, *73*(1), 1–26.
Guiver, J., & Snelson, E. (2009). Bayesian inference for plackett-luce ranking models. In *Proceedings ICML, 26th international conference on machine learning* (pp. 377–384). ACM, ICML '09.
Har-Peled, S., Roth, D., & Zimak, D. (2002a). Constraint classification: A new approach to multiclass classification. In *Proceedings ALT, 13th international conference on algorithmic learning theory* (pp. 365–379). Springer.
Har-Peled, S., Roth, D., & Zimak, D. (2002b). Constraint classification for multiclass classification and ranking. In S. Becker, S. Thrun, & K. Obermayer (Eds.), *Advances in neural information processing systems* (Vol. 15, pp. 809–816). Cambridge: MIT Press.
Hüllermeier, E., Fürnkranz, J., Cheng, W., & Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, *172*(16), 1897–1916.
Hüllermeier, E., & Vanderlooy, S. (2009). Why fuzzy decision trees are good rankers. *IEEE Transactions on Fuzzy Systems*, *17*(6), 1233–1244.
Hunter, D. R. (2004). MM algorithms for generalized Bradley–Terry models. *Annals of Statistics*, *32*(1), 384–406.
Huybrechts, G. (2016). Learning to rank with deep neural networks. Master's thesis, Ecole polytechnique de Louvain (EPL).
Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093.
Kamishima, T., Kazawa, H., & Akaho, S. (2011). A survey and empirical comparison of object ranking methods. In *Preference learning* (pp .181–201). Springer.
Kanda, J., Soares, C., Hruschka, E. R., & de Carvalho, A.C.P.L.F. (2012). A meta-learning approach to select meta-heuristics for the traveling salesman problem using MLP-based label ranking. In *Proceedings ICONIP, 19th international conference on neural information processing* (pp. 488–495). Springer, Doha, Qatar.
Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, *30*(1/2), 81–93.
Kidwell, P., Lebanon, G., & Cleveland, W. (2008). Visualizing incomplete and partially ranked data. *IEEE Transactions on Visualization and Computer Graphics*, *14*(6), 1356–1363.
Krantz, D. H. (1967). Rational distance functions for multidimensional scaling. *Journal of Mathematical Psychology*, *4*(2), 226–245.
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25, pp. 1097–1105). Red Hook: Curran Associates, Inc.
Kruskal, J. B. (1964). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, *29*(1), 1–27.

Lange, K. (2016). *MM optimization algorithms*. Philadelphia: Society for Industrial and Applied Mathematics (SIAM).

Lange, K., Hunter, D., & Yang, I. (2000). Optimization transfer using surrogate objective functions. *Journal of Computational and Graphical Statistics*, *9*, 1–20.

Larochelle, H., Erhan, D., & Bengio, Y. (2008). Zero-data learning of new tasks. In *Proceedings of the AAAI'08, 23rd national conference on artificial intelligence* (pp. 646–651).

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Genetic algorithms for the traveling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, *13*, 129–170. https://doi.org/10.1023/A:1006529012972.

Lichman, M. (2013). UCI Machine Learning Repository. School of Information and Computer Sciences, University of California, Irvine. http://archive.ics.uci.edu/ml.

Liu, T. (2011). *Learning to rank for information retrieval*. New York: Springer.

Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, *45*(1–3), 503–528.

Luce, R. D. (1959). *Individual choice behavior: A theoretical analysis*. New York: Wiley.

Luce, R. D. (1961). A choice theory analysis of similarity judgments. *Psychometrika*, *26*(2), 151–163.

Luenberger, D. G. (1973). *Introduction to linear and nonlinear programming*. Reading, MA: Addison-Wesley.

Luo, T., Wang, D., Liu, R., & Pan, Y. (2015). Stochastic top-k listnet. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 676–684). Association for Computational Linguistics, Lisbon, Portugal.

Mallows, C. L. (1957). Non-null ranking models. *Biometrika*, *44*(1/2), 114–130.

Marden, J. I. (1995). *Analyzing and modeling rank data*. London: Chapman & Hall.

Maystre, L., & Grossglauser, M. (2015). Fast and accurate inference of Plackett–Luce models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28, pp. 172–180). Red Hook: Curran Associates, Inc.

Menke, J. E., & Martinez, T. R. (2008). A Bradley–Terry artificial neural network model for individual ratings in group competitions. *Neural Computing and Applications*, *17*(2), 175–186.

Menon, A. K., & Elkan, C. (2010a). Dyadic prediction using a latent feature log-linear model. arXiv:1006.2156.

Menon, A. K., & Elkan, C. (2010b). A log-linear model with latent features for dyadic prediction. In *Proceedings of the 2010 IEEE international conference on data mining* (pp. 364–373). IEEE Computer Society, ICDM '10.

Menon, A. K., & Elkan, C. (2010c). Predicting labels for dyadic data. *Data Mining and Knowledge Discovery*, *21*(2), 327–343.

Meulman, J. J., Van der Kooj, A. J., & Heiser, W. J. (2004). *Principal components analysis with nonlinear optimal scaling transformations for ordinal and nominal data. The Sage handbook of quantitative methodology for the social sciences* (pp. 49–72). London: Sage.

Mitchell, M. (1998). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.

Murata, N., Kitazono, J., & Ozawa, S. (2017). Multidimensional unfolding based on stochastic neighbor relationship. In *Proceedings of the 9th international conference on machine learning and computing* (pp. 248–252).

Pahikkala, T., Stock, M., Airola, A., Aittokallio, T., De Baets, B., & Waegeman, W. (2014). A two-step learning approach for solving full and almost full cold start problems in dyadic prediction. In T. Calders, F. Esposito, E. Hüllermeier, & R. Meo (Eds.), *Lecture notes in computer science* (Vol. 8725, pp. 517–532). Springer.

Pahikkala, T., Waegeman, W., Airola, A., Salakoski, T., & De Baets, B. (2010). Conditional ranking on relational data. In *Proceedings ECML/PKDD European conference on machine learning and knowledge discovery in databases* (pp. 499–514). Springer.

Pahikkala, T., & Airola, A. (2016). RLScore: Regularized least-squares learners. *Journal of Machine Learning Research*, *17*(221), 1–5.

Pahikkala, T., Airola, A., Stock, M., De Baets, B., & Waegeman, W. (2013). Efficient regularized least-squares algorithms for conditional ranking on relational data. *Machine Learning*, *93*, 321–356.

Plackett, R. L. (1975). The analysis of permutations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *24*(2), 193–202.

Prechelt, L. (2012). Early stopping: But when? In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade* (pp. 53–67). Springer.

Ribeiro, G., Duivesteijn, W., Soares, C., & Knobbe, A. J. (2012). Multilayer perceptron for label ranking. In *Proceedings ICANN, 22nd international conference on artificial neural networks* (pp. 25–32). Springer, Lausanne, Switzerland.

Rigutini, L., Papini, T., Maggini, M., & Scarselli, F. (2011). Sortnet: Learning to rank by a neural preference function. *IEEE Transactions on Neural Networks*, *22*(9), 1368–1380.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 9.

Schäfer, D., & Hüllermeier, E. (2015). Dyad ranking using a bilinear Plackett–Luce model. In *Proceedings ECML/PKDD—2015, European conference on machine learning and knowledge discovery in databases* (pp. 227–242). Springer, Porto, Portugal.

Schäfer, D., & Hüllermeier, E. (2016). Plackett–Luce networks for dyad ranking. In *Workshop LWDA, "Lernen, Wissen, Daten, Analysen"*. Potsdam, Germany.

Soufiani, H., Parkes, D., & Xia, L. (2014). Computing parametric ranking models via rank-breaking. In *Proceedings of ICML, 31st international conference on machine learning*, Beijing, China.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems (NIPS-1988)* (Vol. 1, pp. 99–106). Los Altos: Morgan Kaufmann.

Trohidis, K., Tsoumakas, G., Kalliris, G., & Vlahavas, I. (2008). Multilabel classification of music into emotions. In *Proceedings of ISMIR 2008, international conference on music information retrieval* (pp. 325–330), Philadelphia, PA, USA.

Tsochantaridis, I., Joachims, T., Hofmann, T., & Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, *6*, 1453–1484.

Tsoumakas, I., & Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, *3*(3), 1–13.

Tucker, L. R. (1960). Intra-individual and inter-individual multidimensionality. In H. Gulliksen & S. Messick (Eds.), *Psychological scaling: Theory and applications* (pp. 155–167). New York: Wiley.

Van Deun, K., Groenen, P., & Delbeke, L. (2005). VIPSCAL: A combined vector ideal point model for preference data. Econometric Institute Report No. EI 2005-03, Erasmus University Rotterdam.

Vembu, S., & Gärtner, T. (2010). Label ranking algorithms: A survey. In J. Fürnkranz & E. Hüllermeier (Eds.), *Preference Learning* (pp. 45–64). New York: Springer.

Weimer, M., Karatzoglou, A., Le, Q. V., & Smola, A. J. (2007). COFI RANK: Maximum margin matrix factorization for collaborative ranking. In J. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in neural information processing systems* (Vol. 20, pp. 1593–1600). Cambridge: MIT Press.

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University, Cambridge, MA.

Zhang, M. L., & Zhou, Z. H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, *18*(10), 1338–1351.

Zhou, Y., Liu, Y., Yang, J., He, X., & Liu, L. (2014). A taxonomy of label ranking algorithms. *Journal of Computers*, *9*(3), 557–565.