UNIVERSITÄT PADERBORN

Fakultät für Elektrotechnik, Informatik und Mathematik

Dyad Ranking with Generalized Plackett-Luce Models

Dirk Schäfer

Dissertation

zur Erlangung des Doktorgrades

der Naturwissenschaften

Dr.rer.nat.

Juni 2018

Abstract

The term "dyad ranking" refers to a new problem setting within preference learning. Dyads are feature vector pairs that need to be ranked by machine learning models.

Existing ranking methods do not deliver good results for dyad ranking, since they do not use all features of the dyads. Therefore, three generalizations of the Plackett–Luce (PL) model, a statistical model for rank data, are introduced: Joint-Feature PL (JFPL) uses joint-feature vector representations for the dyads, i.e. a mapping of a vector pair to a single vector. The bilinear PL model (BilinPL), which takes up the idea of JFPL, specifies the joint-feature map by means of the cross product. Experiments show that BilinPL is superior to existing label ranking methods, because the dyad features improve prediction performance and it can deliver predictions on new labels. The third model, PLNetworks (PLNet), does not require the specification of a joint-feature map but instead learns it. The model is based on a neural network and can capture non-linear relationships among preferences.

Applications of dyad ranking include genetic algorithm recommendations, similarity learning of images, and the configuration of image-processing pipelines using preference-based reinforcement learning. To benefit from the probabilistic information produced by the PL models, two visualization approaches based on multidimensional scaling and unfolding are introduced.

Zusammenfassung

Dyad Ranking ist eine neue Problemstellung innerhalb des Präferenzlernens. Dyaden sind Paare von Merkmalsvektoren, die durch Modelle des Maschinellen Lernens in ein Ranking überführt werden sollen.

Bestehende Ranking Methoden liefern keine zufriedenstellenden Ergebnisse für das Dyad Ranking, weil nicht alle Informationen der Dyaden genutzt werden. Aus diesem Grund werden drei Erweiterungen des Plackett–Luce (PL) Modells, einem statistischen Modell für Rangdaten, vorgeschlagen: Joint-Feature PL basiert auf der Idee, Dyaden auf jeweils einen gemeinsamen Merkmalsvektor abzubilden. In dem bilinearen PL Modell (BilinPL) ist die Abbildung über das Kreuzprodukt zwischen Merkmalsvektorpaaren definiert. Experimente zeigen, dass das BilinPL Modell eine bessere Prädiktionsgüte als Label Ranking Methoden aufweist und Rankings über Labels prädizieren kann, die nicht in den Trainingsdaten vorhanden sind. Das dritte Modell, PLNetworks (PLNet), basiert auf einem Neuronalen Netzwerk und ermöglicht das Erlernen der Repräsentationen von gemeinsamen Merkmalsvektoren.

Die Anwendungen umfassen das Meta-Learning zur Empfehlung von genetischen Algorithmen, das Ähnlichkeitslernen und die Konfigurationsbestimmung von Bildverarbeitungsketten auf Basis des präferenzbasierten Verstärkungslernens. Die probabilistische Eigenschaft der Modelle wurde für zwei neue Visualisierungsmethoden genutzt, die auf der Mehrdimensionalen Skalierung und dem Unfolding basieren.

Contents

Notation, Abbreviations, and Acronyms

Notation

$\boldsymbol{z}$ Dyad, i.e., $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} \in \mathbb{X}$ and $\boldsymbol{y} \in \mathbb{Y}$

$\mathbb{X}$ Feature space (domain), resp., $\mathbb{Y}$ and $\mathbb{Z}$

ρ Ranking

ϱ Collection of dyads / objects

π Permutation, also, σ (in context RL, π refers to *policy*)

$\boldsymbol{e}_i$ One-hot vector, i.e. all entries zero except for the i-th component which is one

$\mathcal{D}$ Dataset

$\mathcal{L}$ Loss function

Λ Finite set of labels, i.e., $\Lambda = \{\lambda_1, \ldots, \lambda_M\}$

$\mathbf{P}$ Probability

$\mathbb{S}_M$ Set of all permutations of the length M

N Number of training examples

$\boldsymbol{x}_k$ The k-th vector within a collection (in bold typeface)

$\boldsymbol{x}_k^{(n)}$ The k-th vector of the n-th collection or sample

x_i The i-th vector component of the vector $\boldsymbol{x}$ (in regular typeface)

$x_{k,i}^{(n)}$ The i-th vector component of the k-th vector $\boldsymbol{x}$ in collection n

$\Phi(\boldsymbol{x}, \boldsymbol{y})$ Joint-feature map, i.e. the vector representation of a dyad

$\bar{\boldsymbol{z}}$ Abbreviation for the joint-feature vector $\Phi(\boldsymbol{x}, \boldsymbol{y})$

Abbreviations and Acronyms

API Approximate Policy Iteration

GA Genetic Algorithm

I.I.D. Independent and Identically Distributed

JF Joint-Feature

LCA Luce Choice Axiom

LTR Learning To Rank

MDS Multidimensional Scaling

ML Machine Learning

MLE Maximum Likelihood Estimation

MM Majorization–Minorization

NLL Negative Log-Likelihood

NN Neural Network

PL Plackett–Luce Model

ROL Rank Ordered Logit

RL Reinforcement Learning

RUM Random Utility Model

SGD Stochastic Gradient Descent

ZSL Zero-Shot Learning

1 Introduction

1.1 Preference Learning

Preference learning is an emerging subfield of machine learning (ML) in which problem settings and methods about intelligent processing of preferences are studied and developed (Fürnkranz and Hüllermeier, 2010, 2016).

Object ranking is one such preference learning setting (Cohen et al., 1999; Kamishima et al., 2011). It is about learning a ranking function by observing a subset of ranked objects. The goal is to utilize a learned ranking function by applying it on a set of new objects to find a meaningful ranking for them. An important aspect of the problem is that each object is represented in terms of features or attributes. There is no differentiation between where, when, or who would state the object rankings in this setting and all the rankings are assumed to belong to the global context.

This aspect is different in another preference learning problem called **label ranking** (Fürnkranz and Hüllermeier, 2003; Vembu and Gärtner, 2010; Zhou et al., 2014). It is about learning a "label ranker," which, given a new instance and a finite set of class labels, can be used to predict a ranking over the labels. Before that, it is trained using training examples, where each example consists of an instance—i.e., a collection of features and an associated ranking over a finite set of labels. An instance can be considered as a context that would be added to a ranking of labels. Depending on the context, the set of labels can be ranked differently.

Both settings, namely object and label ranking, seem to have commonalities as well as mutual shortcomings. They share the property of using feature descriptions. In object ranking, the objects are described by feature vectors, whereas in label ranking only the instances are expressed in terms of features. However, the first problem lacks the ability to express object rankings under varying contexts, while the second problem is restricted to dealing with class labels instead of objects. These mutual shortcomings in the two

settings are the main motivation for a more flexible alternative by focusing on using feature vectors.

A categorization of machine learning and also of preference learning can be made in terms of three types of learning: **unsupervised**, **supervised**, and **reinforcement learning** (Goodfellow et al., 2016; Murphy, 2012; Shalev-Shwartz and Ben-David, 2014). In a nutshell, with unsupervised learning one may aim to find and visualize structures within a dataset such as groups of similar items. In contrast to that, in supervised learning one aims to learn a relation between the description of an object and a target structure, which, for example, can be a class label or a real value. The learning of a relation from a finite amount of data should be done in such a way that it becomes possible to predict targets on new objects in the best possible manner. Reinforcement learning is different from the two former types. There, an agent must take decisions on actions within an environment. An agent may learn from delayed feedback, since not every single action is accompanied with an immediate response from which an agent can learn from.

From this point of view, the supervised learning type is predominantly used in this thesis. Reinforcement learning will also play a role, but in a lesser capacity and always in connection with supervised learning.

1.2 Toward a new Problem Setting

A new setting that claims to express and to extend two existing preference learning settings must capture commonalities, but should also exhibit new properties. A common element in label and object ranking displays the fact that feature vectors describe objects in object ranking and instances in label ranking. A way to extend object ranking could be the attempt to introduce contexts in a similar fashion as instances existing in label ranking. They could be represented as features vectors in the same way as instances in label ranking. Likewise, a possibility to extend label ranking could be the attempt to introduce feature vectors in the same way as objects exist in object ranking. This could be accomplished by describing labels in terms of feature vectors just as objects. Hence, a new setting, which offers feature descriptions for both contexts and labels, would extend both the existing settings in a **unifying way**.

Introducing such a unification requires a new and consistent nomenclature. Instead of speaking of contexts, instances, labels, and objects, we may use rather abstract terms. To this end, let us define, on the one hand, the domain $\mathbb{X}$ with the feature vectors $\boldsymbol{x}_i$, which could be related to contexts or instances. And on the other hand, let us define the domain $\mathbb{Y}$ with the feature vectors $\boldsymbol{y}_j$, which could be related to labels and objects. As an illustration, the vectors of the hitherto defined domains could be arranged in a two-dimensional schema (Figure 1.1). Moreover, the vectors of the $\mathbb{X}$ domain are arranged as the rows, and the vectors of the $\mathbb{Y}$ domain are arranged as the columns.

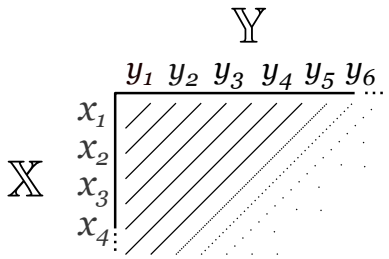

Figure 1.1: Feature vectors arranged in a two dimensional schema.

It needs to be clarified what the term ranking refers to in the new setting. By reflecting on the recently developed extended label ranking and object ranking scenario, it is clear that the feature vectors of one domain are not considered in isolation, but they are always related to the feature vectors from another domain. This means that feature vector pairs are essential elements in the new setting that constitute rankings. We will subsequently refer to a pair of feature vectors as a **dyad**. And dyads are the elements from which the rankings are prepared.

Pairs also play a central role in dyadic prediction, an existing framework that covers collaborative filtering and link prediction as problem instances (Hofmann et al., 1999; Menon, 2013; Menon and Elkan, 2010b; Pahikkala et al., 2014). A key characteristic in the framework is the possibility to address items by identifiers (IDs) and also optionally by feature vectors called side-information. In collaborative filtering, the data consists of ratings that users provided on items, e.g., on movies. One is interested there in methods that can be used to predict ratings on items not yet rated by users. In link prediction, the pairs are items of the same domain, e.g., pairs of people or pairs of molecules. The problem there is to learn the affinities between pairs of such items and to predict hitherto unknown links between new pairs.

In a sense, the new setting can be considered as a transfer of dyadic prediction to the realm of preference learning. To enable a similar flexibility as dyadic prediction, however, it should be allowed in the new setting to consider pairs over objects that are from the same domain as in link prediction. Since the ranking of feature vector pairs is the central aspect and the fact that it resembles dyadic prediction, we will refer to the new setting as **dyad ranking**.

1.3 Research Questions

With the introduction of dyad ranking as a new preference learning setting, there are several questions that come along with it:

- What are the key properties and advantages of dyad ranking compared to the existing preference learning settings?
- More concretely, what is the advantage of considering label attributes rather than pure class labels as in label ranking?
- In which (formal) way does dyad ranking provide a unifying view of the existing preference learning problems such as label and object ranking?
- Which other similar settings are out there and how do they differ from each other?
- What kind of tasks can be expressed and solved with dyad ranking?
- On the methodological level, which methods are suitable for solving dyad ranking tasks?
- What are the strengths and weaknesses of the various dyad ranking methods?
- What are the applications of dyad ranking?

1.4 Outline of the Thesis

The thesis is structured as follows. In Chapter 2, the dyad ranking setting is introduced formally. This includes the definition of the basic elements of dyad ranking and the problem statement. Next, different prediction tasks and dyad types are identified. In Chapter 3, several related settings and methods are described and compared with dyad ranking.

The chapter that follows, Chapter 4, is about dyad ranking methods that address the different dyad ranking tasks. The methods are introduced under a common methodological framework based on joint-feature representations of dyads. The background of the approaches covers discrete choice models and the Luce's choice axiom.

Applications of dyad ranking include the ranking of genetic algorithms, the visualization of dyad rankings using extensions of multidimensional scaling and unfolding, similarity learning, and preference-based reinforcement learning. These are provided in Chapters 5–7.

1.5 Contributions

Several publications resulted during the creation of this thesis. These are

- 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, Springer, Porto, Portugal, pp 227-242 (Schäfer and Hüllermeier, 2015a)

- Schäfer D., Hüllermeier E. (2015) Preference-Based Meta-Learning using Dyad Ranking: Recommending Algorithms in Cold-Start Situations. In: Workshop MetaSel at PKDD/ECML (Schäfer and Hüllermeier, 2015b)

- Schäfer D., Hüllermeier E. (2016) Preference-based Reinforcement Learning using Dyad Ranking. In: Proceedings DA2PL/EURO mini conference, DA2PL (From Multiple Criteria Decision Aid to Preference Learning) (Schäfer and Hüllermeier, 2016b)

- Schäfer D., Hüllermeier E. (2016) Plackett-Luce networks for dyad ranking. In: Workshop LWDA, Lernen, Wissen, Daten, Analysen, Potsdam, Germany (Schäfer and Hüllermeier, 2016a)

- Schäfer D. (2017) A Latent-Feature Plackett-Luce Model for Dyad Ranking Completion In: Workshop LWDA, Lernen, Wissen, Daten, Analysen, Rostock, Germany (Schäfer, 2017)

- Schäfer D., Hüllermeier E. (2018) Dyad Ranking using Plackett-Luce models based on joint-feature representations. In: Machine Learning (Schäfer and Hüllermeier, 2018)

Parts of the bilinear Plackett–Luce model and the content of the chapters 2, 3 and 4.3 are published in (Schäfer and Hüllermeier, 2015a). This also includes, to some extent, the experiments from the sections 7.1.1 and 7.1.2. The section 7.2.3 about cold-start situations in preference-based meta-learning was used in the publication (Schäfer and Hüllermeier, 2015b). The discussion on the relationship between the bilinear Plackett–Luce model and matrix factorization from Section 4.3.2 was picked up in the publication (Schäfer, 2017).

The publication (Schäfer and Hüllermeier, 2016a) was a result of what is described about the Plackett-Luce networks in Chapter 4.4. The contents of Section 4.2 on the Joint-Feature Plackett–Luce model and on the optimization with majorization–minorization in Section 4.3.3, as well as Chapter 5 on multi-dimensional unfolding with dyad ranking and parts of the experiments in Chapter 7 were published in (Schäfer and Hüllermeier, 2018). The outcome of the chapters 6 and 6.4 on preference-based reinforcement-learning was used, to a large extent, in the publication (Schäfer and Hüllermeier, 2016b).

2 Dyad Ranking

Dyad ranking is a preference learning setting, in which rankings of dyads play a central role. The chapter is divided into four parts, where the basic concepts of the setting are introduced first. This is followed by a formulation of dyad ranking as a supervised learning problem. In Section 3, various prediction tasks are identified and described on the basis of a two-dimensional schema. Finally, the key properties of the dyad ranking setting are summarized in Section 4.

2.1 Basic Concepts

There are several elements in dyad ranking, and we begin with the definition of a dyad. A dyad is a pair of feature vectors. To put it more formally, a dyad is specified as a tuple of two feature vectors

$$\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{Z} = \mathbb{X} \times \mathbb{Y} \ , \tag{2.1}$$

which are from two domains $\mathbb{X}$ and $\mathbb{Y}$ respectively, where $\mathbb{X} = \mathbb{Y}$ is not excluded.

The dimensionality and the components of feature vectors can vary and depend on the problem at hand. For the feature vectors, we will use the following notation subsequently:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_r)^\top \in \mathbb{X} = \mathbb{X}_1 \times \mathbb{X}_2 \times \cdots \times \mathbb{X}_r \text{ and} \tag{2.2}$$

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_c)^\top \in \mathbb{Y} = \mathbb{Y}_1 \times \mathbb{Y}_2 \times \cdots \times \mathbb{Y}_c \ . \tag{2.3}$$

A domain $\mathbb{X}_i$ can, for example, be $\mathbb{R}$, $\mathbb{N}$ or $\{0, 1\}$, where the latter would lead to a finite domain $\mathbb{X}$, if r (or c) are finite.

How does a problem or a method influence the choice of features? The properties of objects that are relevant to solving a (prediction) problem have been represented as a collection of features. In this sense, the kinds of features and the selection thereof depend both on the problem and the method to solve the problem. A feature can either

be learned by means of an ML method or be constructed manually. In the latter case, a researcher or an engineer has to decide which among several attributes of an object should be chosen. This requires knowledge of the problem and the method to be used. A feature is typically a number of a certain scale, such as nominal, ordinal, interval or ratio (Stevens, 1946), and describes a certain aspect of an object. It can also be either informative or uninformative. In the first case, a feature contains information which can be used to make predictions on novel dyads. In contrast, a feature of the second kind does not offer such a possibility. It is then merely an identifier, just like a class label in multi-class classification. If an object is already described by several numbers, it is often of importance how to transform these numbers so that they can applied on ML methods. An example is the scaling of values to a certain interval, such as $[0, 1]$ or $[-1, 1]$. Which of those decisions would be appropriate is part of a process called feature engineering. An alternative to feature engineering is the learning of features—the so-called *representation learning* (Bengio et al., 2013).

Next, we will turn to rankings over the previously mentioned dyads. The notation for a ranking over a finite number of dyads is given by

$$\rho : \boldsymbol{z}_{\pi(1)} \succ \boldsymbol{z}_{\pi(2)} \succ \ldots \succ \boldsymbol{z}_{\pi(M)} \ , \tag{2.4}$$

where ρ is an identifier for a ranking and the length is indicated by the number M. Equivalently, a ranking (2.4) can be stated as a pair $\rho = (\varrho, \pi)$, which consists of a finite collection ϱ of dyads and a permutation. The kind of collections we consider are sets with an associated index set $I = [M] = \{1, 2, \ldots, M\}$ respectively, where M may vary across different ranking observations. A permutation $\pi \in \mathbb{S}_M$ is used to address dyads within a collection ϱ so that $\pi(i)$ can be interpreted as the index j of a dyad within ϱ put on the rank position i. Otherwise, the notation $\pi^{-1}(j)$ expresses the rank position of the j-th dyad in the collection $\varrho = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_M\}$. To ease readability, it is possible to reorder the indices of the dyads within a collection ϱ so that they match the special permutation $\pi = (1, 2, \ldots, M)$, where $\pi(i) = i$ for $1 \leq i \leq M$. Doing it that way, the permutation notation in formula (2.4) can be omitted and it is possible instead to write

$$\rho : \boldsymbol{z}_1 \succ \boldsymbol{z}_2 \succ \ldots \succ \boldsymbol{z}_M \ . \tag{2.5}$$

These notations can be used to represent dyad rankings in a generic way. There are two special kinds of dyad rankings which are of practical relevance. We cover these dyad rankings next.

The first is a ranking where one of its members is fixed across a ranking, e.g. where all dyads in a ranking are of the form $\boldsymbol{z}_j = (\boldsymbol{x}, \boldsymbol{y}_j)$; in this case, (2.5) can also be written as

$$\rho : \boldsymbol{y}_1 \succ_{\boldsymbol{x}} \boldsymbol{y}_2 \succ_{\boldsymbol{x}} \ldots \succ_{\boldsymbol{x}} \boldsymbol{y}_M \ . \tag{2.6}$$

This form of ranking is called a *contextual* dyad ranking, and it corresponds to the main type of ranking considered in the thesis.

Another special case is a dyad ranking of length two. This is referred to as a *pairwise (dyadic) preference* and has the form $(\boldsymbol{x}_i, \boldsymbol{y}_i) \succ (\boldsymbol{x}_j, \boldsymbol{y}_j)$, which may also be in a contextualized form. Pairwise preferences can be generated in an unequivocal way from a dyad ranking of the length $M > 2$ by converting it into $\binom{M}{2} = \frac{M(M-1)}{2}$ pairwise preferences. Pairwise comparisons are especially appealing in subjective preference judgements (David, 1969) and hence can be motivated from this point of view. Moreover, the case of pairwise preferences has been studied quite extensively within ML because it reduces the problem of ranking to simpler learning problems such as binary classification (Dekel et al., 2004).

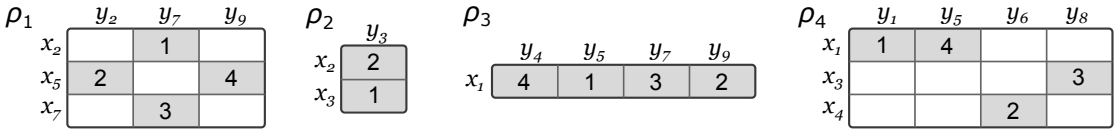

Figure 2.1: Example of a dyad ranking data set $\mathcal{D}$ that contains four rankings $\rho_1 - \rho_4$.

An example of a small dataset that comprises various dyad ranking types is illustrated in Figure 2.1. It picks up the concept of a two-dimensional schema introduced in Chapter 1 and the dataset is denoted by $\mathcal{D} = \{\rho_1, \rho_2, \rho_3, \rho_4\}$. The ranking ρ_1 can be identified from the figure as $\rho_1 : (\boldsymbol{x}_2, \boldsymbol{y}_7) \succ (\boldsymbol{x}_5, \boldsymbol{y}_2) \succ (\boldsymbol{x}_7, \boldsymbol{y}_7) \succ (\boldsymbol{x}_5, \boldsymbol{y}_9)$. The rankings ρ_2 and ρ_3 are both contextual rankings, and ρ_2 differs from the latter in that it is a pairwise ranking.

2.2 Problem Statement

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

$$h : \mathcal{Z} \longrightarrow \mathbb{S}$$

that can be used to predict a ranking $\hat{\pi}$, given a collection of dyads ϱ as an input, where $\mathcal{Z} := \mathcal{P}(\mathbb{Z})$ is the power set of $\mathbb{Z}$ and $\mathbb{S}$ is the set of permutations over the natural numbers. These numbers are used to indicate the dyads within a dyad collection $\varrho \in \mathcal{Z}$.

More specifically, the problem is to seek a model with optimal prediction performance corresponding to finding a risk (expected loss) minimizer

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \int_{\mathcal{Z} \times \mathbb{S}} \mathcal{L}(h(\varrho), \pi) \, d\mathbf{P} \ , \tag{2.7}$$

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

A common choice for the loss $\mathcal{L}$ is a distance metric on rankings. Such a distance metric can be used to measure the quality of a prediction with regard to a ground truth. Some examples are the Kendall tau distance (Kendall, 1938), the Spearman distance, and the Spearman footrule. The Kendall tau distance is defined as the number of discordant pairs:

$$D(\sigma, \pi) = \#\big\{(i, j) \,|\, 1 \leq i < j \leq K, \, (\sigma(i) - \sigma(j))(\pi(i) - \pi(j)) < 0\big\} \ . \tag{2.8}$$

Related to it is the Spearman distance, which is the sum of the squared rank distances

$$D(\sigma, \pi) = \sum_{i=1}^{M} (\sigma(i) - \pi(i))^2 \ ,$$

which used for the definition of the Spearman's rank correlation coefficient, which is $1 - D(\sigma, \pi)/(M(M^2 - 1))$ (Spearman, 1904). The Spearman footrule distance is, in contrast, based on the sum of the absolute differences

$$D(\sigma, \pi) = \sum_{i=1}^{M} |\sigma(i) - \pi(i)| \ .$$

The above-mentioned distance measures are related in the sense that there exist tight bounds among them (Diaconis and Graham, 1977). Thus, optimizing for one of them results in similar performances for the other measures.

Up to this point, we have seen the definition of the risk minimization principle, a principle from statistical learning theory, that explains what we aim for when we want to learn a dyad ranker. However, the joint measure $\mathbf{P}$ in (2.7) is typically unknown in practical applications in such a way that a dyad ranker is learned from training data instead. The training data is assumed to be a finite sample from an unknown joint probability distribution. This reroute is referred to as empirical risk minimization, which corresponds to seeking a dyad ranker, thus minimizing

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(h(\varrho_n), \pi_n) \ ,$$

using a (finite) training set $\mathcal{D}_{tr} = \{(\varrho_n, \pi_n)\}_{n=1}^{N} = \{\rho_n\}_{n=1}^{N}$. A dyad ranker is thus a result of a supervised learning process.

2.3 Prediction Tasks

The goal of the learning process is to produce a dyad ranker that can be used to make ranking predictions on new dyads. Depending on the existence of dyads during the training phase, there are different regions within $\mathbb{X} \times \mathbb{Y}$ identifiable on which predictions can be carried out.

Two subsets within $\mathbb{X}$ and $\mathbb{Y}$ must be considered respectively in order to identify them. Let $\mathbb{X}^{[\text{tr}]}, \mathbb{X}^{[\text{te}]} \subset \mathbb{X}$ with $\mathbb{X}^{[\text{tr}]} \cap \mathbb{X}^{[\text{te}]} = \emptyset$. And similarly, let $\mathbb{Y}^{[\text{tr}]}, \mathbb{Y}^{[\text{te}]} \subset \mathbb{Y}$ with $\mathbb{Y}^{[\text{tr}]} \cap \mathbb{Y}^{[\text{te}]} = \emptyset$. The definitions of $\mathbb{X}^{[tr]}$ and $\mathbb{Y}^{[tr]}$ depend on the existence of dyads within the training set $\mathcal{D}$, the set of dyad rankings used to train a model. We define the set $\mathbb{X}^{[\text{tr}]}$ as a subset of the feature vectors $\boldsymbol{x}$ that are contained in the training set, i.e., $\mathbb{X}^{[\text{tr}]} = \left\{\boldsymbol{x} \in \mathbb{X} \,|\, (\boldsymbol{x}, \boldsymbol{y}) \in \Omega^{[\text{tr}]}\right\}$, where $\Omega^{[\text{tr}]} = \{(\boldsymbol{x}, \boldsymbol{y}) \in \varrho_n \,|\, \varrho_n \in \mathcal{D}\}$. And likewise, we define the set $\mathbb{Y}^{[\text{tr}]}$ as the set of feature vectors $\boldsymbol{y}$ that are contained in the training set, i.e., $\mathbb{Y}^{[\text{tr}]} = \left\{\boldsymbol{y} \in \mathbb{Y} \,|\, (\boldsymbol{x}, \boldsymbol{y}) \in \Omega^{[\text{tr}]}\right\}$. In a similar way, the sets $\mathbb{X}^{[\text{te}]}$ and $\mathbb{Y}^{[\text{te}]}$ can be defined on the basis of a test set.

Then, four regions within $\mathbb{X} \times \mathbb{Y}$ can be identified with these definitions (see Figure 2.2). Each of those regions corresponds to the Cartesian product between two of the sets $\mathbb{X}^{[\text{tr}]}$, $\mathbb{X}^{[\text{te}]}$, $\mathbb{Y}^{[\text{tr}]}$ and $\mathbb{Y}^{[\text{te}]}$. The division also enables the identification of four basic types associated with the regions (see Table 2.1). Although Region 1 consists of dyads from the training set $\mathcal{D}$, this region is typically larger and exhibits feature vector combinations that are not part of the training set.

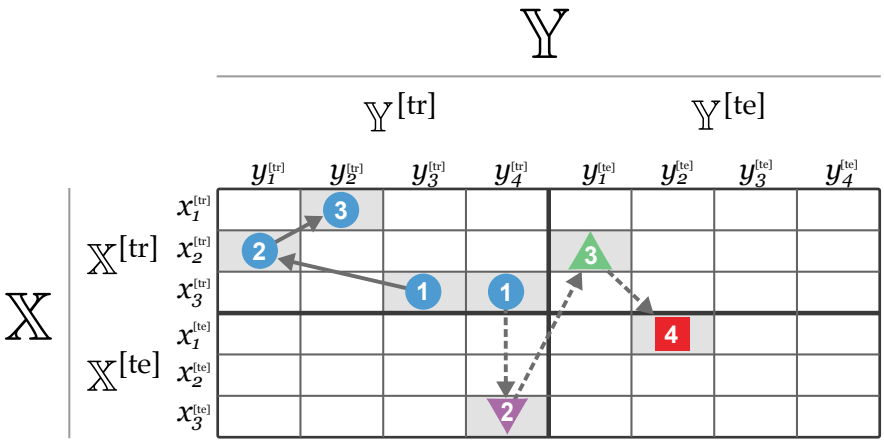

Figure 2.2: Dyads are represented as symbols (circle, upright triangle, upside-down triangle, or square) within a rectangular schema. Dyad rankings are expressed as a sequence of symbols connected with the directed edges. The predictions for the rankings of new dyads can take place in up to four areas and mixtures of those. An example for a predicted dyad ranking is indicated as a sequence with dashed edges. It spans over all four areas.

Table 2.1: Categorization of dyads into four basic types. They differ in the way their feature vectors belong to subsets within their domains.

Type 1	Type 2	Type 3	Type 4
$\mathbb{X}^{[tr]} \times \mathbb{Y}^{[tr]}$	$\mathbb{X}^{[te]} \times \mathbb{Y}^{[tr]}$	$\mathbb{X}^{[tr]} \times \mathbb{Y}^{[te]}$	$\mathbb{X}^{[te]} \times \mathbb{Y}^{[te]}$

We go on by formulating typical prediction tasks involving dyadic data known in literature (Pahikkala et al., 2014; Park and Marcotte, 2012; Stock et al., 2018). Four main tasks can be identified as an analogy to the dyad types. Again, Figure 2.2 is useful for explaining the different tasks. The first prediction task can be formulated as the task of predicting the rankings of dyads within Region 1. The new dyads involved there consists of novel feature vector combinations, where the feature vectors were encountered individually by the learning algorithm at other Type 1 dyads. The second task is about the prediction on dyads where the $\boldsymbol{y}$ feature vectors are known, but the $\boldsymbol{x}$ feature vectors are completely new. Diametrical to this is the third task in which the $\boldsymbol{y}$ vectors are new, but the $\boldsymbol{x}$ are known. The fourth task is about dyads containing feature vector members that are completely new.

These are the main tasks. However, since the predicted rankings can contain mixtures of different dyad types, further tasks are conceivable. These include six tasks where the predicted rankings span over two regions, four further tasks over three regions, and one over all four regions. An example of the latter scenario is indicated in Figure 2.2 as a ranking of dyads with dashed directed edges.

2.4 Key Properties

Dyad ranking is a supervised learning setting that comprises different prediction tasks. The procedure of supervised learning and the prediction in dyad ranking is summarized in Figure 2.3. A learning algorithm, also referred to as the learner, produces a dyad ranking model, which can, in turn, be used to solve one of previously described prediction tasks. It is assumed in the thesis that all the rankings processed by the learner are without ties.

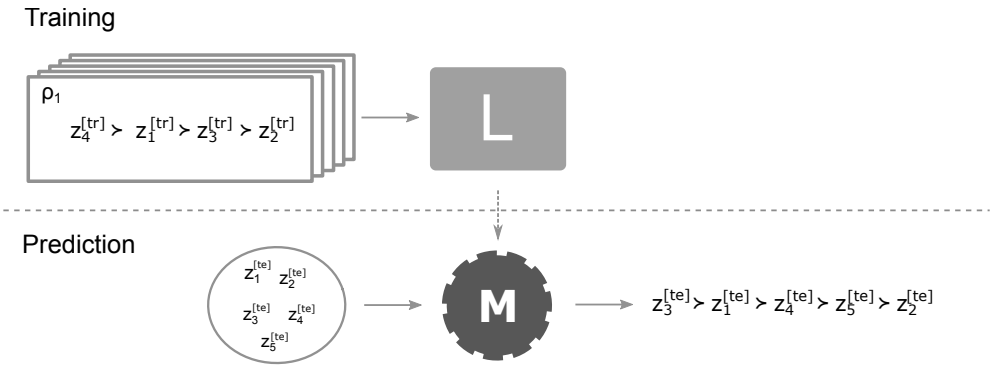

Figure 2.3: Dyad ranking two-step procedure. The output of a learning algorithm at the training phase is a model (a dyad ranker). This can, in turn, be used at the prediction phase for predicting the rankings of a given set of dyads.

One key property of the setting is that there is no a priori restriction on the length of input and output rankings. Although the main focus is on the case where the feature spaces consist of vectors, other types of spaces are also conceivable such as matrices or tensors. Another key property of dyad ranking is that two domains are involved to determine the features of the dyads. The two domains can be different, in which case they are denoted by $\mathbb{X}$ and $\mathbb{Y}$, or they can be identical, in which case they are denoted

both by $\mathbb{X}$. In the case of two identical domains, dyad ranking methods may use the order imposed by the tuple or treat them symmetrically., i.e. $f(\boldsymbol{x}_i, \boldsymbol{x}_j) = f(\boldsymbol{x}_j, \boldsymbol{x}_i)$.

3 Related Settings and Methods

This chapter provides an overview of the settings related to dyad ranking. Each setting is compared with dyad ranking at two levels, namely at the problem description level and at the methodological level. The methods are chosen either because they share some properties with the dyad ranking methods introduced later in Chapter 4, or they are well established for a particular setting according to the literature. The problems discussed in this chapter can be classified roughly in accordance with whether they are dealing with *ranking* (R) or *dyadic* (D) data. According to these criteria, we will discuss those which are provided in Table 3.1:

Table 3.1: Overview of the related settings.

Chapter	Setting	Type
3.1	Label Ranking	D,R
3.2	Object Ranking	R
3.3	Learning To Rank (in Information Retrieval)	D,R
3.4	Collaborative Filtering	D
3.4	Collaborative Ranking	D,R
3.5	Conditional Ranking	R
3.6	Dyadic Prediction	D
3.7	Zero-Shot Learning	D

All of these have in common that they can be explained on the basis of a two-dimensional schema. There are, of course, further well-known settings, such as multivariate regression and multi-class or multi-label classification, whose components could be arranged in such a schema as well. These are excluded from this chapter because they are inherently covered by some of the introduced settings.

3.1 Label Ranking

In label ranking, there are instance vectors with the associated rankings over a set of class labels. The goal in this setting is to learn a label ranker, which, given a new instance vector, is capable of predicting a ranking (total order) over labels (Fürnkranz and Hüllermeier, 2010a; Vembu and Gärtner, 2010; Zhou et al., 2014).

Formally, in label ranking there is an instance space, $\mathbb{X}$, that contains feature vectors and a finite set of labels $\Lambda = \{\lambda_1, \ldots, \lambda_M\}$. A training dataset, $\mathcal{D}_{tr} = \{\rho_i\}_{i=1}^N$, is a set of label rankings, where each label ranking ρ consists of an instance vector $\boldsymbol{x}$ with an associated ranking over the labels. A ranking ρ is described by

$$\lambda_{\pi(1)} \succ_{\boldsymbol{x}} \lambda_{\pi(2)} \succ_{\boldsymbol{x}} \ldots \succ_{\boldsymbol{x}} \lambda_{\pi(M)} \ , \tag{3.1}$$

where $\pi(i)$ is the index of the j-th label in the set Λ put on rank position i. Otherwise, $\pi^{-1}(j)$ denotes the rank position of the label with the index j.

A label ranker is a mapping $\mathbb{X} \to \mathbb{S}_M$, i.e. a mapping from the input space $\mathbb{X}$ to the output space $\mathbb{S}_M$, which is the set of all permutations of the label set Λ. The learning can be described with these notations as a mapping

$$h : \mathbb{X} \longrightarrow \mathbb{S}_M \ ,$$

which can be used for prediction later on. Finding a model with an optimal prediction performance is, therefore, the central part of the problem. It corresponds to finding a risk (expected loss) minimizer

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \int_{\mathbb{X} \times \mathbb{S}_M} \mathcal{L}(h(\boldsymbol{x}), \pi) \, d\mathbf{P} \ , \tag{3.2}$$

over a joint-measure $\mathbf{P} = \mathbf{P}(\boldsymbol{x}, \pi) = \mathbf{P}(\boldsymbol{x})\mathbf{P}(\pi \,|\, \boldsymbol{x})$. The underlying hypothesis space in (3.2) is denoted by $\mathcal{H}$ and the loss function on $\mathbb{S}_M$ is denoted by $\mathcal{L}$. Having obtained the model h^*, a ranking function can then be realized by the setting $\hat{\pi} = h^*(\boldsymbol{x})$ and then labels are arranged using $\hat{\pi}$, as it is done in case of example ranking (3.1).

The prediction task can be stated as follows: Given a new instance vector $\boldsymbol{x}$, apply the learned label ranker on $\boldsymbol{x}$ to obtain a ranking over the finite set of labels.

The setting can be described in terms of a two-dimensional schema in a similar vein as dyad ranking. This can be seen by arranging the instance vectors as rows and the labels as columns (c.f. Figure 3.1). Moreover, by doing it that way, it is possible to

identify the prediction task to be of Type 2 according to Table 2.1. This means that the labels are already known from the training phase while, the instances are new in the prediction phase. One crucial difference with dyad ranking, however, is that the labels in label ranking are not described by any features.

	λ_1	λ_2	λ_3	λ_4
x_1	3		1	2
x_2	1	3	2	4
x_3		1		2
x_4	?	?	?	?
x_5	?	?	?	?

Figure 3.1: Label ranking problem represented in a two-dimensional schema. In this example, the label ranking is associated with $\boldsymbol{x_1}$ is $\lambda_3 \succ \lambda_4 \succ \lambda_1$. What is sought here are the label rankings for the new instance vectors $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$.

We will now go on with the discussion of methods that have been developed to solve the label ranking problem. To give an overview, the three considered methods are *Constrained Classification* (CC), *Ranking By Pairwise Comparison* (RPC) and *Linear Plackett-Luce* (LinPL). The CC and LinPL methods share properties with the dyad ranking methods in terms of learning utility scores. Being based on the probabilistic Plackett–Luce model, the LinPL approach is closely related to the dyad ranking methods discussed in Chapter 4. RPC shares an idea with CC about tackling a complex problem by using binary base models, but it differs from all other approaches in terms of learning utility values for labels. The RPC method is considered because it is well established and has a sound theoretical foundation. Two further neural network-based label ranking approaches are discussed later on in Chapter 4.4 in conjunction with the PLNet approach for dyad ranking.

3.1.1 Constrained Classification

Constrained classification (CC) is rather a framework than an approach tailored specifically to label ranking (Har-Peled et al., 2002a,b). It is possible with CC to tackle various other problems such as binary, multi-class, and multi-label classification. This is accomplished in CC by considering each training example in terms of an instance vector $\boldsymbol{x} \in \mathbb{R}^r$ together with a set of constraints. A single constraint specifies the relative order

of two labels for a particular training example—this is akin to, in terms of label ranking, the expression of a ranking over M labels specifically as a new training example with $M-1$ constraints. In this way, the label ranking $\lambda_3 \succ_x \lambda_1 \succ_x \lambda_2$ would be a training example of the form $\{(\lambda_3 \succ_x \lambda_1), (\lambda_1 \succ_x \lambda_2)\}$). More generically, with a slight change of notation, a label ranking

$$\rho : \lambda_{\pi(1)} \succ_x \cdots \succ_x \lambda_{\pi(M)}$$

is expressed in the CC terms as

$$\left(\boldsymbol{x}, \left\{(\lambda_{\pi(i)} \succ \lambda_{\pi(i+1)}) \,|\, 1 \leq i \leq M_n - 1\right\}\right) \ .$$

CC proceeds from linear (sorting) functions

$$f_i(\boldsymbol{x}) = \sum_{k=1}^{r} w_{ik}^{\top} \cdot x_k = \langle \boldsymbol{w}_i, \boldsymbol{x} \rangle, \ \ 1 \leq i \leq M \ \ , \tag{3.3}$$

with label-dependent weights $\boldsymbol{w}_i$. A preference statement $\lambda_i \succ_x \lambda_j$ can be expressed with (3.3) as a positive constraint $f_i(\boldsymbol{x}) - f_j(\boldsymbol{x}) > 0$ and a negative constraint $f_j(\boldsymbol{x}) - f_i(\boldsymbol{x}) < 0$. The assumption in CC is that functions produce utility scores in $\mathbb{R}$ which can be used for the definition of a linear sorting classifier

$$h(\boldsymbol{x}) = \underset{i=1\ldots M}{\text{argsort}} \ f_i(\boldsymbol{x}) \ \ . \tag{3.4}$$

The classifier (3.4) returns for a given $\boldsymbol{x}$ a permutation π of the natural numbers 1 to M, which can, in turn, be used to express a label ranking:

$$\lambda_{\pi(1)} \succ_x \lambda_{\pi(2)} \succ_x \cdots \succ_x \lambda_{\pi(M)} \ \ .$$

To learn from this kind of information, a transformation step takes place on the thus expressed data. A training example with the vector $\boldsymbol{x} \in \mathbb{R}^r$ and M' pairwise constraints will result, after the transformation, in the vector $\boldsymbol{z} \in \mathbb{R}^{rM'}$ with associated binary classes. More specifically, the instance $\boldsymbol{x}$ with the positive constraint $f_i(\boldsymbol{x}) - f_j(\boldsymbol{x}) > 0$ corresponds to the transformed vector $\boldsymbol{z}$, where the vector $+1 \cdot \boldsymbol{x}$ is copied into the components $((i-1) \cdot r + 1), \ldots, (i \cdot r)$, and $-1 \cdot \boldsymbol{x}$ is copied into the components $((j-1) \cdot r + 1), \ldots, (j \cdot r)$, whereas the remaining components of $\boldsymbol{z}$ are padded with zeros (an expansion otherwise known as *Kesler's construction* for multi-class classification (Duda et al., 2000; Nilsson, 1965)). Similarly, the vector $\boldsymbol{z}$ can be constructed for a negative

constraint using reversed signs. These vectors form positive and negative examples respectively and can then be fed into a linear *binary classifier* such as a support vector machine that aims to find a separating hyperplane $h(\boldsymbol{z}) = \langle \boldsymbol{v}, \boldsymbol{z} \rangle$ in $\mathbb{R}^{M'r}$. To solve the original problem, the weight portion $\boldsymbol{v}_i$ associated with the label i must be extracted from $\boldsymbol{v}$ and can then be used for the definition of $f_i(\boldsymbol{x}) = \langle \boldsymbol{v}_i, \boldsymbol{x} \rangle$. These, in turn, can be used to form a sorting classifier (3.4).

Algorithm 1 CC Online Algorithm for Label Ranking

Require: Input $\mathcal{D} = \{(\boldsymbol{x}_n, \{(\lambda_{\pi_n(i)}, \lambda_{\pi_n(i+1)}) \,|\, 1 \leq i \leq M_n - 1\})\}_{n=1}^{N}$, $\alpha-$bound

repeat
  Initialize $\alpha \in \mathbb{R}^{N \times M \times M}$ with zeros.
  for $n = 1 : N$ **do**
    for $i = 1 : M_n - 1$ **do**
      $f_i = \langle \boldsymbol{w}_{\pi(i)}, \boldsymbol{x}_n \rangle$
      $f_{i+1} = \langle \boldsymbol{w}_{\pi(i+1)}, \boldsymbol{x}_n \rangle$
      if $f_i \leq f_{i+1} \wedge \alpha_{n,i,i+1} \leq \alpha-$bound **then**
        $\alpha_{n,i,i+1} = \alpha_{n,i,i+1} + 1$
        $\boldsymbol{w}_{\pi_n(i)} = \boldsymbol{w}_{\pi_n(i)} + \boldsymbol{x}_n$ ▷ Promote
        $\boldsymbol{w}_{\pi_n(i+1)} = \boldsymbol{w}_{\pi_n(i+1)} - \boldsymbol{x}_n$ ▷ Demote
      end if
    end for
  end for
until Convergence
$h(\boldsymbol{x}) := \underset{i=1\ldots M}{\text{argsort}} \ \langle \boldsymbol{w}_i, \boldsymbol{x} \rangle$
return h

An alternative approach to the problem is an online algorithm proposed by Har-Peled et al. (2002a); this is based on the perceptron with an $\alpha-$bound to make it more robust against noise (Khardon and Wachman, 2007). The perceptron would converge in the case where the preference information is without contradictions noise-free—this, however, is not often the case in reality. Therefore, the $\alpha-$ bound limits the number of updates on preference examples that are not consistent with the hypothesis class. The pseudo-code of the online variant of CC is provided in Listing 1.

3.1.2 Ranking By Pairwise Comparison

Ranking by pairwise comparisons (RPC) belongs to the class of *learning by pairwise comparison* (LPC) methods, which are based on two steps: (i) decomposition of preference information and (ii) aggregation of learned results (Fürnkranz and Hüllermeier, 2003, 2010b; Hüllermeier et al., 2008).

For the decomposition step, the label ranking $\lambda_{\pi(1)} \succ_x \lambda_{\pi(2)} \succ_x \ldots \succ_x \lambda_{\pi(M)}$ over M labels under a context $\boldsymbol{x}$ can be decomposed into $K = M(M-1)/2$ many pairwise preferences $\{\lambda_{\pi(1)} \succ_x \lambda_{\pi(2)}, \ldots, \lambda_{\pi(1)} \succ_x \lambda_{\pi(M)}, \ldots, \lambda_{\pi(M-1)} \succ_x \lambda_{\pi(M)}\}$. A (binary) classification model, $\mathcal{M}_{i,j}$, can then be trained on each of them so that it produces the value 1 if $\lambda_i \succ_x \lambda_j$ or the value zero if otherwise $\lambda_j \succ_x \lambda_i$. Alternatively, instead of producing outputs in $\{0, 1\}$, it is also possible to use (soft) binary classifiers that are capable of producing values in $[0, 1]$.

A trained model $\mathcal{M}_{i,j}$ can later be used to assign a valued (fuzzy) preference relation to any example $\boldsymbol{x}$ by

$$\mathcal{R}_x(\lambda_i, \lambda_j) = \begin{cases} \mathcal{M}_{i,j} & \text{if } i < j, \\ 1 - \mathcal{M}_{i,j} & \text{if } i > j \ . \end{cases}$$

The preference relation is then used for the aggregation step. This step can be realized by using a simple voting strategy (Fürnkranz and Hüllermeier, 2010b; Hüllermeier and Fürnkranz, 2004). A score for the label λ_i is obtained by $S(\lambda_i) = \sum_{\lambda_j \neq \lambda_i} \mathcal{R}_x(\lambda_i, \lambda_j)$. A ranking of labels can then be realized from these scores by arranging them in accordance with the sorting of the scores in descending order, i.e.,

$$(\lambda_i \succeq_x \lambda_j) \Longleftrightarrow (S(\lambda_j) \geq S(\lambda_j)) \ .$$

This voting approach is justified theoretically as it maximizes the expected Spearman rank correlation between a true and a predicted label ranking (Hüllermeier et al., 2008) under the assumption that the binary classifiers provide correct probability estimates, which correspond to

$$\mathcal{R}_x(\lambda_i, \lambda_j) = \mathcal{M}_{i,j}(\boldsymbol{x}) = \mathbf{P}(\lambda_i \succ_x \lambda_j) \ .$$

It has been discussed in (Hüllermeier et al., 2008) that with RPC it is in principle possible to optimize other loss functions such as Kendall's tau distance, but it requires an adoption of the ranking procedure in the aggregation step of the approach.

3.1.3 Linear Plackett–Luce Model for Label Ranking

A further method for label ranking based on the Plackett–Luce model was proposed in (Cheng et al., 2010). The main idea behind this approach is to re-parameterize the original model by label specific functions that take the instance vectors $\boldsymbol{x}$ into account. This is accomplished by replacing the original Plackett–Luce model parameters $\boldsymbol{v} = (v_1, \ldots, v_K)$ with label–dependent (log-linear) functions $v_i = v_i(x)$,

$$v_i = v_i(\boldsymbol{x}) = \exp\left(\sum_{k=1}^{d} w_{i,k} \cdot x_k\right) = \exp\left(\langle \boldsymbol{w}_i, \boldsymbol{x} \rangle\right) \quad , \tag{3.5}$$

where each function represents a utility score for the i-th label. The actual model parameters become the label–dependent weight vectors $\boldsymbol{w}_i$ in Equation (3.5), which can be found via maximum likelihood estimation (MLE). The index $1 < i < K$ in $w_{i,j}$ corresponds to the i-th label out of K labels (or "alternatives") and the index $1 < k < d$ refers to the k-th dimension.

Since the linear Plackett–Luce model is similar to the generalized Plackett–Luce approaches for dyad ranking in Chapter 4, the Plackett–Luce model and the MLE procedure will be described in more detail there.

Given the estimates of these parameters, the 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 (3.5), and a ranking $\hat{\pi}$ is determined by sorting the labels λ_i in decreasing order of their (predicted) values $\hat{v}_i$. The ranking $\hat{\pi}$ is a reasonable prediction, as it corresponds to the mode of the distribution $\mathbf{P}(\cdot \,|\, \hat{\boldsymbol{v}})$.

3.2 Object Ranking

Object ranking (or *learning to order things*) is a preference learning setting in which the rankings of objects are of central importance (Cohen et al., 1999; Fürnkranz and Hüllermeier, 2010; Kamishima et al., 2011). An object, in contrast to a label, has attributes and is described in terms of a feature vector.

	o_1	o_2	o_3	o_4	o_5	o_6
ϱ_1	3	2		1		
ϱ_2			1	2		
ϱ_3	2	3	1			
ϱ_4			?	?	?	
ϱ_5	?			?		?

Figure 3.2: Object ranking problem represented in a two-dimensional schema. In the example, the objects for training are the sets ϱ_1, ϱ_2 and ϱ_3. They are accompanied by incomplete rankings over the objects $\boldsymbol{o}_1$-$\boldsymbol{o}_4$. What is sought here are the objects rankings for the sets $\varrho_4 = \{\boldsymbol{o}_3, \boldsymbol{o}_4, \boldsymbol{o}_5\}$ and $\varrho_5 = \{\boldsymbol{o}_1, \boldsymbol{o}_4, \boldsymbol{o}_6\}$.

The goal is to find a *ranking function* that accepts example rankings over a subset $\mathbf{O} \subset \mathbb{O}$ of objects as input, where $\mathbb{O}$ 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 $\mathbf{O}$. More formally, the learning problem consists of learning a model

$$h : \mathcal{P}(\mathbb{O}) \longrightarrow \mathbb{S}$$

that can be used to predict the ranking $\hat{\pi}$, given a collection of objects $\varrho = \{\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots\}$ as an input. More specifically, the problem is to find a model with optimal prediction performance which corresponds to seeking an expected loss minimizer

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \int_{\mathcal{P}(\mathbb{O}) \times \mathbb{S}} \mathcal{L}(h(\varrho), \pi) \, d\,\mathbf{P}(\varrho, \pi) \ , \tag{3.6}$$

where $\mathcal{H}$ is the underlying hypothesis space and $\mathcal{L}$ is a loss function on $\mathbb{S}$. Having obtained the model h^*, a ranking function can then be realized straightforwardly by using the prediction $\hat{\pi} = h^*(\varrho)$ on ϱ to arrange its objects by $\boldsymbol{o}_{\hat{\pi}(1)} \succ \boldsymbol{o}_{\hat{\pi}(2)} \succ \cdots \succ \boldsymbol{o}_{\hat{\pi}(|\varrho|)}$. The loss functions that can be used here correspond to those that have already been used in dyad and label ranking.

The training information consists of a set of incomplete rankings $\mathcal{D}_{tr} = \{\rho_n\}_{n=1}^{N}$. This means that the rankings $\rho_n = (\varrho_n, \pi_n)$ can be of different lengths and thus cover different subsets $O_n \in \mathcal{P}(\mathbb{O})$ of objects. This point is illustrated in Figure 3.2.

The object and dyad ranking settings differ in some aspects, but they also have some commonalities. Both have a similar goal: to obtain a ranker that can be applied on sets of unordered objects. From this point of view, a dyad in dyad ranking could be considered as a special kind of structured object. And in this regard, dyad ranking could be considered as an extension of object ranking with dyads as special kinds of objects. The other way around, object ranking can be seen as dyad ranking where all training rankings are observed in a common neutral context. In this respect, it is straightforward to apply dyad ranking methods to object ranking problems, too, namely by equating the "object space" $\mathbb{O}$ with the "dyad space" $\mathbb{Z}$. In this analogy, the prediction task would be of Type 3 if the rankings with their respective neutral contexts are aligned at the rows and the objects are aligned at the columns of the two-dimensional schema in Figure 2.2. The other way around, if a suitable representation of dyads can be found in terms of single objects, object ranking methods could be used for dyad ranking.

We will proceed with the methods that have been developed for object ranking. Cohen's method is primarily chosen because it belongs to the seminal paper that defined the object ranking problem. The second method, expected rank regression, is a well-established approach for the object ranking setting. It is based on the Thurstone model (Case V) from which a connection can be drawn with the methods we propose for dyad ranking. Both methods are also cited in the survey article on object ranking from Kamishima et al. (2011).

3.2.1 Cohen's Method

Cohen's method consists of two steps (Cohen et al., 1998, 1999). In the first step, a preference function, $\mathrm{PREF}(\boldsymbol{o}_i, \boldsymbol{o}_j)$, is learned; it indicates whether to rank the object $\boldsymbol{o}_i$ before $\boldsymbol{o}_j$. And in the second step, a greedy ordering algorithm is applied for finding a ranking of objects that maximally agrees with the preference function PREF. The main assumption underlying Cohen's method is the existence of object features that are of the ordinal type; this is a rather restrictive assumption and limits the application of that approach severely.

The preference function PREF to be learned is composed of K preference functions R_{f_k} in the following way:

$$\text{PREF}(\boldsymbol{o}_i, \boldsymbol{o}_j) = \sum_{k=1}^{K} w_k R_{f_k}(\boldsymbol{o}_i, \boldsymbol{o}_j) \ . \tag{3.7}$$

The learning of weights $\boldsymbol{w} = (w_1, \ldots, w_k)$ takes place in the first step, while the second step is required later to obtain the actual rankings. The basic preference function R_{f_k} is induced by an *ordering function* f using the following definition:

$$R_{f_k}(\boldsymbol{o}_i, \boldsymbol{o}_j) = \begin{cases} 1 & \text{if } f_k(\boldsymbol{o}_i) > f_k(\boldsymbol{o}_j) \\ 0 & \text{if } f_k(\boldsymbol{o}_i) < f_k(\boldsymbol{o}_j) \\ 0.5 & \text{otherwise} \quad . \end{cases} \tag{3.8}$$

The ordering functions f_k are feature extraction methods on the objects. Each of them produces a real value that can be used for comparison. The use case provided in the original papers by Cohen et al. is about combining the results of several searches performed on a domain-specific search engine. More specifically, the functions f_k are associated with a certain search aspect, and they produce higher numbers for the documents listed at higher top rank positions. Kamishima et al. (2011) propose to generalize this by using the attributes of the objects in a more direct way where the ordering functions are defined as $f_k(\boldsymbol{o}_i) = o_{ik}$.

The procedure for learning the preference function PREF is given in Listing 2, in which the loss function is defined as

$$\text{Loss}(R_k, \rho) = \frac{\sum_{\boldsymbol{o}_i \succ \boldsymbol{o}_j \in \rho} (1 - R_k(\boldsymbol{o}_i, \boldsymbol{o}_j))}{|\rho|} \ . \tag{3.9}$$

The weight allocation algorithm is based on the *Hedge algorithm* (Freund and Schapire, 1997) and to some extent on the *weighted majority algorithm* (Littlestone and Warmuth, 1994).

The second step involves the calculation of PREF according to Eq. (3.7) on a set of new objects. What follows is the application of a greedy order algorithm to produce a ranking that agrees best with the PREF function. We will not go further into details at this point and refer to the original publications.

Algorithm 2 Weight Allocation Algorithm

Require: Input $\mathcal{D} = \{\rho_n\}_{n=1}^{N}$, β parameter $\in [0, 1]$ (default: 0.9)

Initialize weights $w_k = \frac{1}{K}$
Create ordering functions $f_k,\ 1 \leq k \leq K$
repeat
    for $n = 1$ **to** N **do**
        Create basic preference functions R_k using ϱ_n
        Evaluate losses $\text{Loss}(R_k,\, \rho_n)$ ▷ See equation (3.9)
        Update $w_k = w_k \cdot \beta^{\text{Loss}(R_k, \rho_n)}$
        Normalize $w_k = w_k / \sum_{k=1}^{K} w_k$
    end for
until Convergence
return $\boldsymbol{w}$

3.2.2 Expected Rank Regression

Expected ranking regression (ERR) is an approach that can be combined with standard regression techniques for learning and predicting the expected ranks (Kamishima et al., 2005; Kamishima et al., 2011).

The assumption underlying ERR is that the learner is faced with a set of incomplete object rankings ρ_n and that each ranking ρ_n has a complete unobservable counterpart ρ_n^* from which it is generated by deleting objects randomly according to the uniform distribution. More specifically, it is assumed that the object rankings are generated in accordance with the Thurstone model (Case V) (Marden, 1995; Thurstone, 1927). Objects are arranged there in ascending order of the scores $f^*(\boldsymbol{o})$ given by the normal distribution, i.e. $f^*(\boldsymbol{o}) \sim \mathcal{N}(\mu(\boldsymbol{o}), \sigma^2)$, where $\mu(\boldsymbol{o})$ is the mean in terms of a function and σ is the standard deviation.

ERR makes use of the fact that the conditional expectation of the rank of an object $\boldsymbol{o}_m$ of an underlying ground truth ranking, ρ_n^*, given the incomplete ranking ρ_n, can be calculated as

$$\mathbb{E}[r(\rho_n^*, \boldsymbol{o}_m) \,|\, \rho_n] \propto \frac{r(\rho_n, \boldsymbol{o}_m)}{|\rho_n| + 1} \ , \tag{3.10}$$

where $r(\rho_n, \boldsymbol{o}_m)$ denotes the rank of object $\boldsymbol{o}_m$ within the ranking ρ_n. The learning can thus be done, for example, with multiple linear regression on a training set, which is of

the form

$$\mathcal{D} = \left\{ \left(\boldsymbol{o}_i, \frac{r(\rho_n, \boldsymbol{o}_i)}{|\rho_n| + 1} \right) \right\}_{n=1}^{N} ,$$

where $1 \leq i \leq |\rho_n|$. The outcome is the regression function f with the learned weight $\boldsymbol{w}$, which can be used to calculate scores on a set of new objects, e.g. with $f(\boldsymbol{o}_i) = \langle \boldsymbol{w}, [1, o_{i1}, \ldots, o_{ic}] \rangle$. These scores can in turn be used to create rankings by arranging the objects in ascending order of the scores.

3.3 Learning To Rank

Learning to rank (LTR[1]) is the application of machine learning for information retrieval (Liu, 2011; Mohri et al., 2012). There is not just a single setting in LTR, but rather a variety of settings that exhibit some similarities to those covered in this chapter. Consequently, the methods also bear some resemblance.

	d_1	d_2	d_3	d_4	d_5	d_6
q_1	0	2		2		
q_2	2	2				
q_3		1	0	1		
q_4	?	?	?	?	?	?
q_5	?	?	?	?	?	?

Figure 3.3: Learning to rank problem represented in a two-dimensional schema. In this example, training data consists of relevance degrees that are associated with query-document pairs, where the degrees 2, 1 and 0 mean *definitely*, *possibly*, and *not relevant* respectively. A global document ranking for a given (new) query is sought.

The common theme in all LTR settings can be illustrated with a two-dimensional schema, as it is provided in Figure 3.3. It consists of queries, documents, and an incomplete matrix of preference judgments. The ultimate goal in LTR is to find a method that can be used to provide a global ranking of documents for a given query. From the learning point of view, the problem consists of learning from queries and the relevance of documents with regard to them. Depending on the type of relevance judgments, it is common in LTR to group learning algorithms into three kinds of approaches: the pointwise, the pairwise, and the listwise approaches. Methods of the latter group require the preference of documents in the form rankings. Pairwise approaches, in contrast, are adopted to deal with the pairwise preference on documents. And pointwise approaches can be used if the ground truth labels consist of the relevance degrees about single documents.

We are going to proceed with the formulation of LTR in terms of a risk minimization problem. Let $\mathbb{X}$ be the space of the feature vectors $\boldsymbol{x} = \Phi(q, d)$, which are created using the feature extraction functions Φ from query-document pairs. These functions

[1]Other abbreviations used in the literature are L2R or LeToR.

are specific to information retrieval, and it is common in LTR literature to assume that these vectors already exist. Again, we denote with $\varrho = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots\}$ a collection of vectors and with $\pi \in \mathbb{S}$ a permutation. The goal in LTR can then be stated as the search for a model that minimizes

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \int_{\mathcal{P}(\mathbb{X}) \times \mathbb{S}} \mathcal{L}(h(\varrho), \pi)\, d\,\mathbf{P}(\varrho, \pi) \ , \tag{3.11}$$

where $\mathbf{P}(\boldsymbol{x}, \pi) = \mathbf{P}(\boldsymbol{x})\mathbf{P}(\pi \,|\, \boldsymbol{x})$, $\mathcal{H}$ is the underlying hypothesis space and $\mathcal{L}$ is a loss function on $\mathbb{S}$. With this kind of problem formulation, we focus on the pairwise and listwise approaches only. This is justifiable since they are closer to the approaches that we study in dyad ranking.

Dyad ranking and LTR share similar concepts. One of the crucial concepts they share is the relevance of preferences with regard to object **pairs**. And similarly, the preferences are also the subject of learning in a supervised way. Moreover, if queries are seen as the vectors $\boldsymbol{x}$ aligned at the rows and the documents seen as the vectors $\boldsymbol{y}$ aligned at the columns of a two-dimensional schema, LTR is particularly close to what we consider as *contextual* dyad ranking. The queries in LTR could be considered as the contexts $\boldsymbol{x}$ for preference statements over the documents $\boldsymbol{y}$. This correspondence becomes even more distinct by considering the listwise approaches. They seem to be very close to what is needed to solve dyad ranking problems as well because they require preference information in the form of rankings as input and also produce rankings as output.

There are, however, crucial differences with dyad ranking. To begin with, the methods in LTR are tailored to the information retrieval application. More concretely, the users of a search engine are interested in good retrieval results at top positions of a ranking, whereas the quality of an ordering deeper down the list is not much of a concern. This impacts the way methods are constructed, learned, and evaluated. Evaluation measures on rankings like mean average precision (MAP) (Baeza-Yates and Ribeiro-Neto, 1999) or normalized discounted cumulative gain (NDCG) (Järvelin and Kekäläinen, 2000, 2002) put higher emphasis on the top positions. This is a clear difference with dyad ranking where such a constraint or specialization is not prescribed. A further difference between LTR and dyad ranking is the way features are constructed. In LTR, queries and documents are usually not considered individually in terms of feature vectors, but there are single feature vectors that represent query-document pairs. These features are, for

example, constructed by using the term frequencies of query words in the documents and outputs of models like BM25 (Robertson et al., 2009) or PageRank (Brin and Page, 1998).

As for the pairwise approaches, we will have look at RankNet and the support vector machine adopted for the ranking problem, and all of these are well-established methods. The ranking function of the latter is linear with respect to the weight vector and is thus similar to the one used for the joint-feature Plackett–Luce model for dyad ranking in Chapter 4.2. As for the listwise approaches, we will exemplarily inspect ListMLE and ListNet. They are related to the dyad ranking approaches because they are based on the Plackett–Luce model as well. ListNet and RankNet are discussed later in detail with PLNet, a neural network-based approach for dyad ranking in Chapter 4.4.

3.3.1 Ranking Support Vector Machine

We will use *RankSVM* subsequently as a term for two similar approaches that are known in the literature as *Support Vector Ordinal Regression* (SVOR) (Herbrich et al., 1998) and *Ranking SVM* (Joachims, 2002; Joachims and Radlinski, 2007). The problem of ordinal regression was adapted in the Ranking SVM approach specifically for the scenario of information retrieval.

The aim of RankSVM is to learn a linear ranking function of the form $\boldsymbol{w}^\top \Phi(q, d)$, where $\boldsymbol{w}$ is a weight to be learned from data and ϕ is a feature extractor. Joachims (2002) proceeds from the learning problem description (3.11), where *rankings* on query-document pairs are taken as training information. The negative Kendall's tau measure is used there as the loss function $\mathcal{L}$, and it is learned there in a pairwise fashion by minimizing the number of discordant ranking element pairs. This, in turn, is equivalent to determining the weight vector $\boldsymbol{w}$, which satisfies the maximum number of the following inequalities:

$$\begin{aligned} \forall\, (q_1, d_i) \succ (q_1, d_j) \in \rho_1 : \boldsymbol{w}^\top \Phi(q_1, d_i) > \boldsymbol{w}^\top \Phi(q_1, d_j) \\ \dots \\ \forall\, (q_n, d_i) \succ (q_n, d_j) \in \rho_n : \boldsymbol{w}^\top \Phi(q_n, d_i) > \boldsymbol{w}^\top \Phi(q_n, d_j) \ . \end{aligned}$$

Finding the weight vector $\boldsymbol{w}$ that satisfies all inequalities is an NP-hard problem[2]. To solve it, at least approximately, the RankSVM approach is based on an SVM with *slack variables* $\xi_{ij}^{(k)}$. The optimization problem with the above-mentioned inequalities as constraints can therefore be stated as a quadratic problem (QP):

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum \xi_{ij}^{(k)} \\ &\text{subject to:} \\ \forall\,(q_1,d_i) \succ (q_1,d_j) \in \rho_1 :\ &\boldsymbol{w}^\top \Phi(q_1,d_i) \geq \boldsymbol{w}^\top \Phi(q_1,d_j) + 1 - \xi_{ij}^{(1)} \\ &\dots \\ \forall\,(q_n,d_i) \succ (q_n,d_j) \in \rho_n :\ &\boldsymbol{w}^\top \Phi(q_n,d_i) \geq \boldsymbol{w}^\top \Phi(q_n,d_j) + 1 - \xi_{ij}^{(n)} \\ &\forall i \forall j \forall k : \xi_{ij}^{(k)} \geq 0\ . \end{aligned}$$

The standard classification SVM can be applied analogously to this problem as well. To this end, it has to be applied on pairs of difference vectors $\Phi(q_k,d_i) - \Phi(q_k,d_j)$. This can be seen when the constraints are rearranged to $\boldsymbol{w}^\top(\Phi(q_k,d_i) - \Phi(q_k,d_j)) \geq 1 - \xi_{ij}^{(k)}$. Solving quadratic programming problems, such as the given one, can be done, for example, with interior point methods (Vanderbei, 1999), or more specifically, using stochastic pairwise descent (Sculley, 2009). With such a learned weight vector, $\boldsymbol{w}$, it is possible to rank a set of documents for a new query, q, by just calculating scores via $\boldsymbol{w}^\top\Phi(q,d_i)$ and sorting them in descending order. Another notable aspect is the possibility of extending this approach to the non-linear case using kernels (Chen et al., 2017).

3.3.2 ListMLE

ListMLE is a listwise LTR approach based on the Plackett–Luce model and a linear neuronal network (Xia et al., 2009a, 2008, 2009b). It is constructed to solve the learning problem (3.11), whose empirical variant is

$$\mathcal{R}_{\text{emp}}(h) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}\left(h(\varrho_n), \pi_n\right)\ , \tag{3.12}$$

for a given training dataset, $\mathcal{D} = \{(\varrho_n, \pi_n)\}_{n=1}^N$, where $\varrho_n = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_{M_n}\}$. Again, a permutation, π, determines the ranking of query-document elements that stem from an

[2] In computational complexity theory, *non-deterministic polynomial-time* refers to a specific class of decision problems.

associated collection, ϱ. The permutations, therefore, provide the training information for supervised learning. Xia et al. (2008) use a permutation level 0-1 loss; it is defined as

$$\mathcal{L}(h(\varrho)) = \begin{cases} 1 & \text{if } h(\varrho) \neq \pi \\ 0 & \text{if } h(\varrho) = \pi \end{cases} . \tag{3.13}$$

The ranking function h is hereby defined with the scoring function g as

$$h(\varrho) = \underset{i=1\ldots M}{\text{argsort}}\ g(\boldsymbol{x}_i) \ . \tag{3.14}$$

To overcome the difficulty in optimizing the non-differentiable 0-1 loss function in (3.13), the authors resort to the *likelihood loss* as a surrogate loss function, which is specified as $\mathcal{L} = -\log P(\pi \mid \varrho; g)$ with

$$P(\pi \mid \varrho; g) = \prod_{i=1}^{M} \frac{\exp\left(g(\boldsymbol{x}_{\pi(i)})\right)}{\sum_{k=i}^{M} \exp\left(g(\boldsymbol{x}_{\pi(k)})\right)} \ ,$$

which, in fact, is a parameterized variant of the Plackett–Luce model.

In ListMLE, the function g is a *linear* neuronal network depicted in Figure 3.4. The

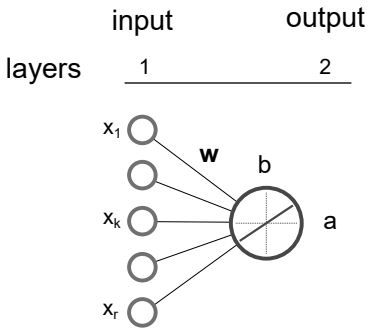

Figure 3.4: ListMLE - linear neuronal network architecture. The network establishes the scoring function g.

output of such a network is $a_i = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b$, and the ranking function g is defined as $g(\boldsymbol{x}_i) = a_i$. The actual learning is performed via stochastic gradient descent, which is summarized in Algorithm 3.

Algorithm 3 ListMLE Training Algorithm

Require: Input $\mathcal{D} = \{(\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{M_n}\}, \pi_n)\}_{n=1}^{N}$, learning rate η, tolerance level ϵ

Initialize weight parameter $\boldsymbol{w}$
repeat
  for $n = 1$ **to** N **do**
    Input $\left(\boldsymbol{x}_{\pi(i)}\right)_{i=1}^{M_n}$ to the linear neural network and compute gradient $\nabla \boldsymbol{w}$
    Update $\boldsymbol{w} = \boldsymbol{w} - \eta \cdot \nabla \boldsymbol{w}$
  end for
  Calculate likelihood loss on the training set $\mathcal{D}$
until Change of likelihood loss is below ϵ
return $\boldsymbol{w}$

3.4 Collaborative Filtering and Ranking

A short overview of collaborative filtering (CF) is provided first and the closely related problem called collaborative ranking (CR) is introduced afterward.

CF is a setting about user preferences on items, and CF methods are used to deliver recommendations for users on items without the need of explicit information about users and items. One of the main drivers for the creation of many CF methods was the Netflix Prize competition in 2006—it explains why many CF methods are typically more application-oriented and tailored to the recommender systems use case. The competition enabled many researchers and practitioners access to large-scale preference data with millions of user ratings for the first time (Koren and Bell, 2015). Additional information on users is typically not made public due to privacy concerns. This may one of the reasons why early CF methods were developed without requiring the features of users and items. However, newer approaches are capable of utilizing those (Bayer et al., 2017; He et al., 2017).

Figure 3.5 shows the elements involved in CF for the case of explicit preference data. Users $U = \{u_1, \ldots, u_N\}$ are arranged at the rows, and the items $I = \{i_1, \ldots, i_M\}$ are located at the columns of a two-dimensional matrix. The cells of the $N \times M$ matrix R consists of the rating values of users ranging from a to b, here 1 to 5, where a higher grade r_{ij} corresponds to a higher preference that a user u_i has for a particular movie i_j. The rating matrix is not fully observable and thus the task is to make statements about the unknown cells.

	i_1	i_2	i_3	i_4
u_1	?	5	3	4
u_2	4	?	?	?
u_3	5	5	?	3
u_4	5	?	5	2

Figure 3.5: Collaborative filtering/ranking problem. In the example, the training data consists of user-item pairs with associated ratings. The rating values for empty cells are sought in case of collaborative filtering. In collaborative ranking, in contrast, recommendations are given more directly in the form of rankings instead—e.g. a ranking over the items i_2, i_3 and i_4 would be the prediction goal for User 2.

The ranking variant of CF is CR. It differs from the former in that the aim is not to find unknown ratings as an intermediate step. The objective instead is the creation of recommendation lists in a direct way. This is accomplished by predicting item rankings for users instead of rating values (Weimer et al., 2008).

Both dyad ranking and CF/CR operate on preferences related to pairs and are of course within the realm of preference learning. CF/CR methods exhibit typically a stronger application-oriented character, for example, the consideration of temporal dynamics of user-item interactions, or the utilization of implicit data from user behavior with a system, or the use of contextual environment information, or the incorporation of counter-measures for fraud (Adomavicius and Tuzhilin, 2015; Hu et al., 2008; Koren, 2010; Rendle et al., 2009; Shi et al., 2014). Another issue has been found with CF/CR in recommender systems—it is the *cold-start problem* (Schein et al., 2002). This refers to the situation in which new users or new items enter a system. A challenge, then, is to provide still meaningful recommendations on new entities. This situation has a counterpart in dyad ranking and is described as a set of tasks involving the ranking of different new types of dyads (i.e. Types 2–4), as described in Chapter 2.3.

From the dyad ranking viewpoint, CF and CR are different because their input types are typically not rankings. Furthermore, CF and CR are concerned with two different domains, which are typically users and items. Dyad ranking is more flexible in this regard, as it allows for handling pairs over arbitrary domains.

As pointed out by Koren and Bell (2015), there are two broad techniques of CF: the

neighborhood and the *matrix factorization approach*[3]. As for CF, we will have a look at PMF (Salakhutdinov and Mnih, 2008), a traditional matrix factorization approach. This approach is chosen because its relation with the bilinear Plackett–Luce model for dyad ranking is discussed with regard to its extensibility as a matrix factorization approach in Chapter 4.3.2. And as for CR, we will concentrate on an approach proposed by Volkovs and Zemel (2012). It is about the engineering of joint-feature representations for dyads using neighborhood techniques. It does not necessarily require explicit features, although their incorporation can be done in a straightforward way. This is, in particular, interesting because such joint-feature representations are the prerequisites to use it together with JFPL, the first model for dyad ranking covered in Chapter 4.2.

3.4.1 Probabilistic Matrix Factorization

Probabilistic matrix factorization (PMF) is a method for collaborative filtering where it is assumed that the rating matrix is of low rank (Salakhutdinov and Mnih, 2008). Furthermore, the rating values r_{ui} of the matrix R are assumed to be either integers ranging from 1 to K or real values. Salakhutdinov and Mnih (2008) propose to adopt a probabilistic linear model with Gaussian observation noise—this is justified by the usage of the root mean squared error (RMSE) evaluation measure on the test set. It is given by

$$\mathrm{RMSE} = \sqrt{\frac{1}{|R_{\mathrm{test}}|} \sum_{r_{ui} \in R_{\mathrm{test}}} (\hat{r}_{ui} - r_{ui})^2} \ , \tag{3.15}$$

where r refers to the ground truth and $\hat{r}$ is a prediction.

Let $f(x \,|\, \mu, \sigma^2)$ be the probability density of the normal distribution with the parameters mean μ and variance σ^2. The conditional distribution over the observed entries of the rating matrix $R \in \mathbb{R}^{N \times M}$ is modeled then as

$$p(R \,|\, \boldsymbol{U}, \boldsymbol{V}, \sigma^2) = \prod_{u=1}^{N} \prod_{i=1}^{M} \Big[f(r_{ui} \,|\, \boldsymbol{U}_{u\cdot}^{\top} \boldsymbol{V}_{i\cdot}, \sigma^2) \Big]^{I_{ui}} \tag{3.16}$$

where $\boldsymbol{U} \in \mathbb{R}^{k \times N}$ and $\boldsymbol{V} \in \mathbb{R}^{k \times M}$ are matrices of k-dimensional latent feature vectors arranged in columns and I_{ui} is an indicator variable, which is 1 if item i has been rated by user u and 0 otherwise. Gaussian priors are furthermore proposed over the matrices

[3] Alternatively referred to as *latent factor model*.

$\boldsymbol{U}$ and $\boldsymbol{V}$ to prevent the over-fitting of the model to the rating matrix. These are given by

$$p(\boldsymbol{U} \mid \sigma_U^2) = \prod_{u=1}^{N} f(\boldsymbol{U}_{u\cdot} \mid 0, \sigma_U^2 \boldsymbol{I})$$
$$p(\boldsymbol{V} \mid \sigma_V^2) = \prod_{i=1}^{M} f(\boldsymbol{V}_{i\cdot} \mid 0, \sigma_V^2 \boldsymbol{I}) \ .$$

Taking the negative log-posterior over user and item features with hyper-parameters $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_V = \sigma^2/\sigma_V^2$ results in the objective

$$E = \frac{1}{2} \sum_{u=1}^{N} \sum_{i=1}^{M} I_{ui}(r_{ui} - \boldsymbol{U}_{u\cdot}^{\top} \boldsymbol{V}_{i\cdot})^2 + \frac{\lambda_U}{2} \|\boldsymbol{U}\|_F^2 + \frac{\lambda_V}{2} \|\boldsymbol{V}\|_F^2 \ , \tag{3.17}$$

which is equivalent to minimizing the sum-of-squared errors with quadratic regularization terms. An evaluation of the model can be accomplished with (3.15) using $\hat{r}_{ui} = \boldsymbol{U}_{u\cdot}^{\top} \boldsymbol{V}_{i\cdot}$ on test user-item pairs. The algorithm for realizing the model is not concretely specified in the original PMF publication (Salakhutdinov and Mnih, 2008). Thus, an implementation could presumably be based on the alternating gradient descent in conjunction with the Adam optimizer (Kingma and Ba, 2014).

3.4.2 Win-Loss-Tie: A Feature Based CR Model

The core idea of the win-loss tie (WTL) approach of Volkovs and Zemel (2012) is the use of LTR methods to tackle the CR problem by extracting appropriate features. It is based on the observation that both CR and LTR are similar if users are considered as queries and items as documents. However, LTR methods typically rely on the existence of the suitable feature representations of query-document pairs. The construction of such features is hence the main aspect of the WTL method.

With a feature representation, it is possible to use any LTR method in WTL. In (Volkovs and Zemel, 2012), it is proposed to evaluate rankings using NDCG and to use LambdaRank (Burges, 2010; Burges et al., 2007) to learn a linear scoring function. The scoring function for the user $u \in U$ and the item $i \in I$ has the following form:

$$f(u, i) = \langle \boldsymbol{w}, \Phi(u, i) \rangle + [U(i) \setminus u = \emptyset] \cdot b_0 \ , \tag{3.18}$$

where $U(i)$ denotes the set of users that rated the item i and $[P]$ is the indicator function. The second summand in (3.18) addresses the case where an item is not rated by any

other users. In this case, the bias b_0 provides a base score and $\Phi(u, i)$ is set to the zero vector.

As requisite for extracting features suitable for LTR methods, WTL utilizes a notion of similarity between user preferences in a similar way as the neighborhood techniques in CF. To measure the similarity between the two users u and v in terms of their ratings, one can use the Pearson correlation coefficient (PPC) (Resnick et al., 1994), which is given by

$$s_p(u,v) = \frac{\sum\limits_{i \in I(u) \cap I(v)} (r_{ui} - \hat{r}_u)(r_{vi} - \hat{r}_v)}{\left[\sum\limits_{i \in I(u) \cap I(v)} (r_{ui} - \hat{r}_u)^2 \sum\limits_{i \in I(u) \cap I(v)} (r_{vi} - \hat{r}_v)^2 \right]^{1/2}} \ , \tag{3.19}$$

where $I(u)$ denotes the set of items that user u has rated and $\bar{r}_u$ refers to the mean rating value across the items $I(u)$. Another common alternative choice is the cosine similarity (Breese et al., 1998), which is given by

$$s_c(u,v) = \frac{\sum\limits_{i \in I(u) \cap I(v)} r_{ui} r_{vi}}{\left[\sum\limits_{i \in I(u) \cap I(v)} r_{ui}^2 \sum\limits_{i \in I(u) \cap I(v)} r_{vi}^2 \right]^{1/2}} \ . \tag{3.20}$$

With such a similarity measure, it is possible to extract further properties from the preference data. Let $K_u(i)$ be the set of the K most similar users of u that rated item i. We can then define three K-dimensional vectors, where each component is defined with $v \in K_u(i)$ as

$$\begin{aligned} \mathrm{WIN}_{ui}(v) &= \frac{1}{|I(v)| - 1} \sum_{j \in I(v) \setminus i} [r_{vi} > r_{vj}] \\ \mathrm{LOSS}_{ui}(v) &= \frac{1}{|I(v)| - 1} \sum_{j \in I(v) \setminus i} [r_{vi} < r_{vj}] \\ \mathrm{TIE}_{ui}(v) &= \frac{1}{|I(v)| - 1} \sum_{j \in I(v) \setminus i} [r_{vi} = r_{vj}] \ . \end{aligned} \tag{3.21}$$

Basic descriptive statistics of these vectors are then extracted and used for the definition of γ-vectors, whose construction is exemplarily shown for the WIN_{ui} vector. The LOSS_{ui} and TIE_{ui} vectors are constructed accordingly. The γ-vector consists of the mean value,

the standard deviation, and the min and the max as follows:

$$\gamma(\text{WIN}_{ui}) = \left[\mu(\text{WIN}_{ui}), \sigma(\text{WIN}_{ui}), \min(\text{WIN}_{ui}), \max(\text{WIN}_{ui}), \frac{1}{|K_u(i)|} \sum_{v \in K_u(i)} [\text{WIN}_{ui}(v) \neq 0]\right] . \tag{3.22}$$

The last component in (3.22) counts positive preference statements toward the item i among the K neighbors of the user u. Finally, the joint-feature representations for the scoring function (3.18) can be defined using the γ-vectors with

$$\Phi(u, i) = [\gamma(\text{WIN}_{ui}), \gamma(\text{LOSS}_{ui}), \gamma(\text{TIE}_{ui}), 1] \quad . \tag{3.23}$$

For the prediction of item rankings for a new user, u, the procedure is as follows. First, for each item $i \in I$, the features $\Phi(u, i)$ are constructed using (3.21), (3.22), and (3.23). Then a ranking of items can be generated using the learned scoring function $f(u, i)$ from Eq. (3.18).

3.5 Conditional Ranking

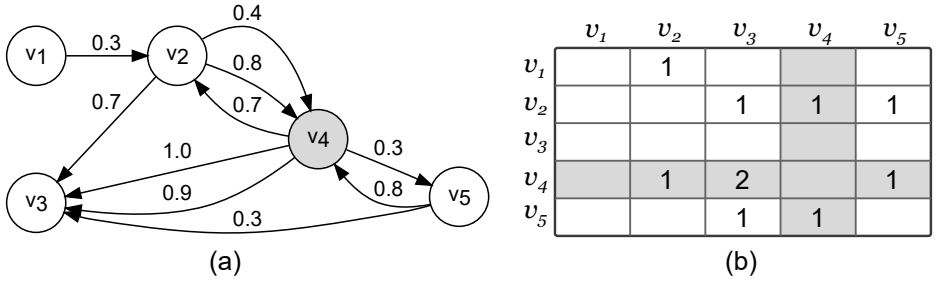

Figure 3.6: Conditional ranking setting. (a) is the training information given in the form of a multi-graph; (b) is the corresponding adjacency matrix.

Conditional ranking is a setting for relational data represented as multi-graphs (Pahikkala et al., 2013, 2010). Nodes in the graph correspond to the objects that are described by features. Directed edges are labeled so as to characterize a binary relation between two nodes. The problem associated with conditional ranking is to rank nodes relative to particular nodes of interest. This can also include new nodes that are not yet part of the graph. Since rankings are always sought to be relative to other nodes, they can be considered as being contextualized or "conditioned" on these target nodes.

Figure 3.6 shows an example of a multi-graph in conditional ranking. It consists of several nodes and edges that are labeled with a real value from the interval $[0, 1]$. An edge corresponds to a single observation, where a higher number expresses a stronger association. Multi-graphs, as opposed to standard graphs, are used in conditional ranking to support multiple observations between two nodes such as those provided in the example between the nodes v_2 and v_4. A ranking of the nodes conditioned on node v_4 can furthermore be created by sorting the edge weights of (v_4, v_2), (v_4, v_3) and (v_4, v_5) in descending order—this results in the ranking $v_3 \succ_{v_4} v_2 \succ_{v_4} v_5$.

Graphs are of central importance in this setting, and depending on the type of edge values, different settings can be considered. Ordinal edge values result in the *instance ranking* setting and continuous edge values resulting in the *object ranking* setting.

In a nutshell, instance ranking is an umbrella term for bipartite and multipartite ranking defined in (Fürnkranz and Hüllermeier, 2010). It is about finding a ranking function that enables the ranking of instances $\boldsymbol{x}$. The training information there consists of instances and classes having a natural order.

A second property of the graphs is the type of relations it expresses. The edges can represent relations that are either symmetric or reciprocal. Symmetric relations lead to graphs with non-directed edges. With real edge values, metric or similarity learning problems can be tackled, while binary values lead to a two-class classification problem. Reciprocal relations are expressed with directed edges, and every edge implicitly introduces a complementary edge so that both edge values add up to one. The domains in which reciprocal relations play a role are, for example, preference learning and bioinformatics.

Formally, the data in conditional ranking is structured as a graph $G = (V, E, Q)$, where V denotes the set of nodes and $E \subset V^2$ identifies the set of edges. It is assumed that the edge labels y_e are generated from a non-observable relation, $Q : V^2 \to [0, 1]$. With Q and a triplet of nodes v, v', and v'', it is possible to state a conditional ranking (or rather preference) as follows

$$v' \succeq_v v'' \Leftrightarrow Q(v, v') \geq Q(v, v'') \ . \tag{3.24}$$

A training set consists of edge label pairs and is denoted as $T = \{(e, y_e) \,|\, e \in E\}$. With a training set, the learning objective can be stated as the minimization of the empirical risk

$$h^* \in \operatorname*{argmin}_{h \in \mathcal{H}} \sum_{v \in V} \sum_{e, e' \in E_v : y_e < y_{e'}} \mathcal{L}(e, e', h) \ , \tag{3.25}$$

where E_v denotes the set of edges associated with node v and the loss function $\mathcal{L}$ is defined as

$$\mathcal{L}(e, e', h) = \begin{cases} 1 & \text{if } h(e) - h(e') > 0 \\ 0.5 & \text{if } h(e) = h(e') \\ 0 & \text{if } h(e) - h(e') < 0 \ . \end{cases} \tag{3.26}$$

The objective (3.25) together with the loss function (3.26) is chosen this way to achieve conditional rankings (3.24) with a learned model h.

The methods for conditional ranking differ from those of LTR in such a way that they exploit only one domain and can thus be more efficient. Another difference is that LTR methods predominantly use similarity relations. Conditional ranking methods, in contrast, are more general in this regard because they can learn from arbitrary binary relations. From an algorithmic point of view, conditional ranking methods could also be used for dyadic relations although they were originally introduced for monadic relations.

On a settings level, it is common for both conditional and dyad ranking to produce predictions in the form of rankings. In most applications so far, however, conditional ranking is concerned with graphs where the nodes stem from a single domain. From the dyad ranking point of view, this is only one possible option. The "conditioning" aspect in conditional ranking with regard to preference relations is the counterpart of a *contextual* dyad ranking. Conditional ranking, however, is fundamentally different and more flexible in terms of handling various types of binary relations between objects. Furthermore, a key difference to conditional ranking is that in dyad ranking the prediction and also the ground truth are rankings.

The canonical method for conditional ranking, described in the seminal conditional ranking paper, is a kernel-based regularized least squares (RLS) method to address various scenarios (Pahikkala et al., 2010). It is related to the bilinear Plackett–Luce model, described in Chapter 4.3, with respect to the joint-feature map it utilizes, which is defined as the Kronecker product.

3.5.1 RankRLS

RankRLS is an approach based on regularized least-squares and was originally invented for LTR acting on pairwise preferences; it has been adopted later for *conditional ranking* (Pahikkala et al., 2007). It also resembles another approach called MPRank (Cortes et al., 2007).

One assumption to solve conditional ranking tasks with RankRLS is that the square loss on edges is a good surrogate for the non-convex loss function (3.26) introduced above. The surrogate loss function is defined as

$$\mathcal{L}(e, e', h) = \left[(y_e - y_{e'}) - (h(e) - h(e'))\right]^2 \quad , \tag{3.27}$$

where the models h in RankRLS are stated in a dual representation, which is

$$h(e) = \langle \boldsymbol{w}, \Phi(e) \rangle = \sum_{e' \in E} a_e K^{\Phi}(e, e') \ . \tag{3.28}$$

The joint feature map Φ in (3.28) refers to a joint-feature map on edges; $\boldsymbol{w}$ and $a_e \in \mathbb{R}$ correspond to parameters to be learned and K^{Φ} is a pairwise kernel. The Kronecker product is defined as

$$\Phi(e) = \Phi(v_a, v_b) = v_a \otimes v_b \ . \tag{3.29}$$

And for the kernels in (3.28), there exist different possibilities, and their choice depend on the assumed type of relation. If no special relation is assumed, then the *Kronecker product pairwise kernel* is used:

$$K^{\Phi}_{\otimes}(e, e') = K^{\Phi}_{\otimes}(v_a, v_b, v'_a, v'_b) = K(v_a, v'_a)K(v_b, v'_b) \ . \tag{3.30}$$

In case of reciprocal relations, the kernel is defined as

$$K^{\Phi}_{\otimes R}(e, e') = \frac{1}{2}(K(v_a, v'_a)K(v_b, v'_b) - K(v_a, v'_b)K(v_b, v'_a)) \ , \tag{3.31}$$

and in a similar way, a symmetric kernel can be expressed by exchanging the minus with the plus sign.

The algorithmic aspects of RankRLS involve the solution of a system of linear equations with a conjugate gradient-based approach using different ways to prevent overfitting, namely the use of Tikhonov regularization and early-stopping (Engl et al., 1996; Evgeniou et al., 2000). The actual implementation is provided in the software package RLScore and is described in (Pahikkala and Airola, 2016).

3.6 Dyadic Prediction

	c_1	c_2	c_3	c_4	c_5	c_6
r_1		5	3	?	?	
r_2	5	?	4	3		?
r_3	5	4		4	?	
r_4		?	?		?	
r_5	?		?	?		?

Figure 3.7: Dyadic prediction represented in a two-dimensional schema. In this example, the training data consists of dyads with associated numerical values (e.g. ratings). Here we seek numerical values for the dyads (r_1, c_1), (r_1, c_4), (r_2, c_2), and (r_3, c_3). If side-information about the dyads is available, it might be possible to make predictions on dyads involving the new rows r_4, r_5 and the new columns c_5, c_6.

Dyadic prediction is a setting which provides a unifying view of *collaborative filtering* and *link prediction* (Menon, 2013; Menon and Elkan, 2010a,c). The key elements in these settings are dyads with associated labels. Training information is given in this kind of form, and the goal is to apply a learned model on new dyads to predict their unknown labels. More specifically, in collaborative filtering the labels are ratings associated with (user, item) pairs, while in link prediction the labels indicate the presence or the absence of an edge between two nodes in a graph.

In contrast to the classification setting, the basic training information in dyadic prediction does not consist of a tuple $(\boldsymbol{x}, y)$, $\boldsymbol{x} \in \mathbb{R}^p, y \in \mathcal{Y}$. Instead, the $\boldsymbol{x}$ in dyadic prediction is a dyad that consists of two entities $\boldsymbol{x} = (r, c)$ with $r \in \mathcal{R}$, $c \in \mathcal{C}$, the so-called dyad members. A dyad member is characterized by having a unique *identifier* and an optional association called *side-information*. The variable names r and c that characterize a dyad are inspired by a rectangular schema in which the dyadic prediction problems can be expressed. An example thereof is given in Figure 3.7.

The way dyads are defined is different from that of dyad ranking. This is owing to the closer relation between dyadic prediction and CF. And as mentioned in Section 3.4, early CF approaches did not take user and item attributes into account. Furthermore, nodes in a link prediction graph are not necessarily described by features. These are the reasons why the dyads in dyadic prediction are composed of numerical IDs. As an

aside, an alternative definition of a dyad for dyadic prediction was used in (Pahikkala et al., 2014), where dyads correspond to instance and task pairs, which are represented in terms of informative *feature vectors*.

Side-information is a concept that also stems from the CF setting. It enables the incorporation of user and item descriptions in the model. The ability of a model using side-information provides a solution to *cold-start* problems (c.f. Section 3.4). In dyadic prediction, the side-information is made up of feature vectors which can either be associated with $r \in \mathcal{R}$, $c \in \mathcal{C}$, or jointly with $\boldsymbol{x} = (r, c)$; it is denoted by $\boldsymbol{s}_r$, $\boldsymbol{s}_c$ and $\boldsymbol{s}_{rc}$ respectively.

In comparison with dyad ranking, the dyadic prediction setting is similar in the sense that it also acts on pair-input data and offers the same flexibility in terms of considering pairs from two domains or from one common domain. Although, as previously mentioned, the definition of a dyad is different, a further difference in dyadic prediction is the output space. Outputs are single values, e.g. ratings, whereas in dyad ranking these are rankings. Dyadic prediction is flexible by supporting different ranges of ordinal rating values, while dyad ranking has a counterpart in supporting rankings of different lengths.

The first mentioned method for dyadic prediction is called *Latent Feature Log-linear Model* (LFL) (Menon and Elkan, 2010a). It is related to the bilinear PL model for dyad ranking in the sense that a top-one PL model with label features resembles a multinomial logistic regression model (c.f. Chapter 4.3), which, in turn, is similar to the formulation taken with LFL .

3.6.1 Latent Feature Log-Linear Model for Dyadic Prediction

The latent feature log-linear model (LFL) (Menon, 2013; Menon and Elkan, 2010a,b,c) is the main approach for dyadic prediction as it provides the flexibility required to address the various options of the setting.

To begin with, LFL can be considered rather as a variety of models, instead of a single model, which are based on the multinomial logistic regression model using latent features. The main assumption underlying these models is that the probability of observing a label does depend log-linearly on the weighted (latent) features. Using the above-introduced notation, the conditional probability of observing the label $y \in \mathcal{Y} = \{1, 2, \ldots, R\}$ for the

dyad $\boldsymbol{x} = (r, c)$ is given as

$$\mathbf{P}(y \mid \boldsymbol{x} = (r,c), \theta_{M_1}) = \frac{\exp\left(\sum_{k=1}^{K} u_{rk}^{(y)} v_{ck}^{(y)}\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} u_{rk}^{(y')} v_{ck}^{(y')}\right)} \ , \tag{3.32}$$

where $\theta_{M_1} = \{\boldsymbol{U}^{(1)}, \dots \boldsymbol{U}^{(R)}, \boldsymbol{V}^{(1)}, \dots, \boldsymbol{V}^{(R)}\}$ are the parameters of the first model. A latent feature, $u_{rk}^{(y)}$, in (3.32) is an element of the label-specific latent feature matrix $\boldsymbol{U}^{(y)}$ with $|\mathcal{R}| \times K$ entries and where K is the dimension of latent feature space. The feature matrix $\boldsymbol{V}^{(y)}$ is similarly defined with $|\mathcal{C}| \times K$ entries.

By adding a constant value of one at the last column of each row of the latent feature matrices, it is possible to model a row, a column, and a label-specific bias at once. The resulting model can be stated as

$$\mathbf{P}(y \mid \boldsymbol{x} = (r,c), \theta_{M_2}) \propto \exp\left(\sum_{k=1}^{K} u_{rk}^{(y)} v_{ck}^{(y)} + a_r^{(y)} + b_c^{(y)} + \mu^{(y)}\right) \ , \tag{3.33}$$

where $\theta_{M_2} = \{\boldsymbol{U}^{(1)}, \dots, \boldsymbol{V}^{(R)}, \boldsymbol{a}^{(1)}, \dots, \boldsymbol{a}^{(R)}, \boldsymbol{b}^{(1)}, \dots, \boldsymbol{b}^{(R)}, \mu^{(1)}, \dots, \mu^{(R)}\}$ and $a_r^{(y)} \in \boldsymbol{a}^{(y)}$ is a row specific, $b_c^{(y)} \in \boldsymbol{b}^{(y)}$ a column-specific and μ^y a label-specific bias.

Side-information can be incorporated into the model in a straightforward way. The resulting model can then be stated as

$$\mathbf{P}(y \mid \boldsymbol{x} = (r,c), \theta_{M_3}) \propto \exp\left(\sum_{k=1}^{K} u_{rk}^{(y)} v_{ck}^{(y)} + (\boldsymbol{w}^{(y)})^\top \bar{\boldsymbol{s}}_{rc}\right) \ , \tag{3.34}$$

using the concatenated side-information vector $\bar{\boldsymbol{s}}_{rc} := [\boldsymbol{s}_r, \ \boldsymbol{s}_c, \ \boldsymbol{s}_{rc}]$ and where the parameters are $\theta_{M_3} = \{\boldsymbol{U}^{(1)}, \dots, \boldsymbol{V}^{(R)}, \boldsymbol{w}^{(1)}, \dots, \boldsymbol{w}^{(R)}\}$.

In link-prediction, the rows and columns belong to the same domain. In the case where the graph is unweighted and undirected, the prediction matrix would be symmetric and binary. Symmetry can be enforced so that $\mathbf{P}(y = 1 \,|\, (r,c)) = \mathbf{P}(y = 1 \,|\, (c,r))$. In this case, the LFL model can be stated (with $y = 0$ as the base class) as

$$\mathbf{P}(y = 1 \mid \boldsymbol{x} = (r,c), \theta_{M_4}) = \frac{\exp\left(\boldsymbol{u}_{r\cdot} \boldsymbol{u}_{c\cdot}^\top\right)}{1 + \exp\left(\boldsymbol{u}_{r\cdot} \boldsymbol{u}_{c\cdot}^\top\right)} \ , \tag{3.35}$$

with $\theta_{M_4} = \{\boldsymbol{U}\}$.

There are different objectives of learning that depend on the type of labels. In the case of nominal labels, Menon and Elkan (2010a) suggests to use the negative conditional log-likelihood with an ℓ_2 regularization term. With the dataset $\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, this is

$$f_{\text{nom}}(\theta) = \frac{\lambda}{2} - \sum_{n=1}^{N} \log \mathbf{P}(y_i \,|\, \boldsymbol{x}_i, \theta) \ . \tag{3.36}$$

If otherwise the labels are of ordinal type, one may use the mean absolute error (MAE), which is $\mathcal{L}(y, \hat{y}) = |y - \hat{y}|$, to reduce the discrepancy between ground truth and prediction. Let the model prediction function be $F(\boldsymbol{x}, \theta)$, and let us define it to be the expectation (mean) $\mathbb{E}[y] = \sum_y y\mathbf{P}(y \,|\, \boldsymbol{x}, \theta)$, then the objective can be stated as

$$f_{\text{ord}}(\theta) = \frac{\lambda}{2} - \sum_{n=1}^{N} |y_i - F(\boldsymbol{x}, \theta)| \ . \tag{3.37}$$

The parameters of the models can be learned with stochastic gradient descent (SGD) (Bottou, 2010) or with L-BFGS (Liu and Nocedal, 1989). A two-step approach is proposed to learn the LFL model with side-information. In the first step, the model is learned with latent features only. And after this, in the second step, the latent features are used to initialize a new model, in which only the weights for the side-information vectors are optimized, while the latent features are kept fix.

With the learned weights, a prediction can be carried out by plugging a dyad and each label $y \in \mathcal{Y}$ respectively into one of the formulas (3.32)-(3.36). The resulting probabilities can then be used to choose the most probable label.

3.7 Zero-Shot Learning

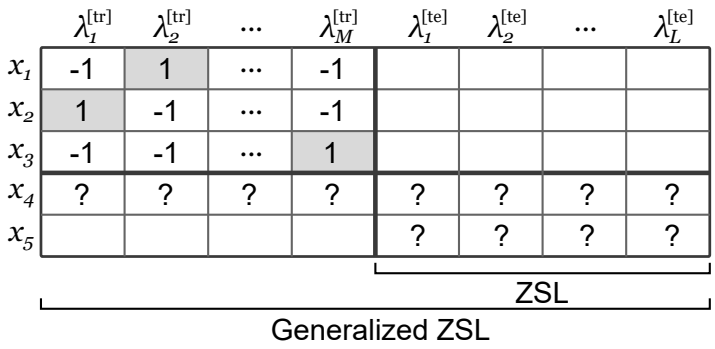

Figure 3.8: Zero-shot learning problem.

Zero-shot learning (ZSL) is a classification setting where predictions are emphasized for class labels that were not observed at training time (Xian et al., 2017). It is the main characteristic that differentiates this from other settings such as standard classification, where the sets comprising training and test class-labels are identical. The instances to be classified in the ZSL literature stem predominantly from the computer vision domain (Frome et al., 2013; Lampert et al., 2014; Mensink et al., 2012). We will concentrate on ZSL in connection with **multi-class classification**, but it should be mentioned that the problem of **multi-label classification** has also gained attention in the ZSL literature (Lee et al., 2017; Sappadla et al., 2016).

The ZSL setting can be further partitioned into two variants, which are the *classical* and the *generalized* setting (Xian et al., 2017). In the former, the predictions are carried out only on those classes that have not been encountered at the training phase, whereas the latter includes predictions over all available class labels that comprise training as well as novel class labels. This difference is also reflected in Figure 3.8, which shows that the problem of ZSL can be described in terms of a two-dimensional schema, where the instances are aligned at the rows and the labels at the columns.

The formal description begins with the definition with the data domains, which are an instance space $\mathbb{X}$ and a label space Λ. The label space is divided into the set of labels $\Lambda^{[\mathrm{tr}]}$ available for training and the set of labels $\Lambda^{[\mathrm{te}]}$ that exist at the test phase. The same distinction can in principle be made on the instance domain too, but this distinction is omitted to keep the notation clear. The training set can, therefore, be stated as $\mathcal{D}^{[\mathrm{tr}]} = \left\{ \left(\boldsymbol{x}_n, \lambda_n^{[\mathrm{tr}]} \right) \right\}_{n=1}^{N}$, where $\lambda_n^{[\mathrm{tr}]} \in \Lambda^{[\mathrm{tr}]}$. The task associated with ZSL can

be stated with these definitions as the learning of a model $h : \mathbb{X} \to \Lambda$ that minimizes the following empirical risk

$$\mathcal{R}_{\text{emp}}(h) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}\left(h(\boldsymbol{x}_n), \lambda_n^{[\text{tr}]}\right) \quad , \tag{3.38}$$

where $\mathcal{L}$ is a loss function for classification. The important point is that the model h should be applicable at the test phase on the test labels $\lambda^{[\text{te}]} \in \Lambda^{[\text{te}]}$, where $\Lambda^{[\text{te}]} \cap \Lambda^{[\text{tr}]} = \emptyset$. The generalized setting, in contrast, requires h to produce sensible results when it is applied to a the larger label space, which is $\Lambda^{[\text{tr}]} \cup \Lambda^{[\text{te}]}$.

Both the ZSL and dyad ranking settings share the aspect of dealing with class labels, resp. $\boldsymbol{y}$ vectors, at the prediction phase that are beyond those which were present at the training phase. In ZSL, the prediction on *new* labels is an inherent part of the setting and therefore a mandatory task. Whereas in dyad ranking there are different tasks, all of which do not necessarily require the prediction of new $\boldsymbol{y}$ vectors. Another difference lies in the output space. It consists of class labels in ZSL, whereas in dyad ranking these are rankings.

Somewhat related is a setting with a similar name, which is called *one-shot learning*. The emphasis to learn classifications has been put on very few examples (Fei-Fei et al., 2006). The aspect of predicting new labels is present in nearly all considered settings so far. The exception to this is label ranking and implicitly multi-class and multi-label classification.

The first ZSL method, which will be described subsequently, is *ESZSL*. It is chosen because its model formulation is based on a bilinear weight matrix. This is also the case in the bilinear Plackett–Luce model for dyad ranking, though there are crucial differences in the learning goal. The second method, *PR*, solves the ZSL task using rankings. It does this by following a probabilistic approach on ranking—this is also the case with all the proposed dyad ranking methods in this thesis including the Plackett–Luce model.

3.7.1 Embarrassingly Simple Approach to Zero-Shot Learning

ESZSL is an approach by Romera-Paredes and Torr (2015) that is similar to that of (Pahikkala et al., 2014) and proceeds from the existing framework introduced in (Akata et al., 2013). The idea there is that the relationship between instance features, class labels, and label attributes can be modeled with two layers. The first layer is learned at

the training stage and establishes a connection between the instance and the class label attributes. The second layer, in contrast, is used to model the relationship between class labels and class label attributes. This second layer is interchangeable because training and test class labels are supposed to be different.

To describe ESZSL formally, the following notation is introduced. The number of training class labels is denoted by $M = \left|\Lambda^{[\mathrm{tr}]}\right|$, and each label consists of c many label attributes. Then, the *class label signature* is a matrix $\mathbb{Y}^{[\mathrm{tr}]} \in [0,1]^{c \times M}$. It contains Boolean or real values in $[0,1]$ that characterize for each attribute its membership to a class label. Let $\mathbb{X}^{[\mathrm{tr}]} \in \mathbb{R}^{r \times N}$ be a matrix of N instance vectors, where each is described by r features. The ground truth classes associated with the instances is given by the matrix $\boldsymbol{C} \in \{-1,1\}^{N \times M}$, in which the case of multiple class memberships per instance vector is possible.

The learning problem consists of learning the weights $\boldsymbol{W} \in \mathbb{R}^{r \times c}$ for minimizing the following objective

$$\min_{\boldsymbol{W}} \mathcal{L}\left(\left(\mathbb{X}^{[\mathrm{tr}]}\right)^{\top} \boldsymbol{W}\, \mathbb{Y}^{[\mathrm{tr}]}, \boldsymbol{C}\right) + \Omega(\boldsymbol{W}) \ , \tag{3.39}$$

where Ω is a regularizer and $\mathcal{L}$ refers to a loss function. The assumption underlying ESZL is that the relationship between instances and attributes can be modeled linearly. The label prediction for a new instance vector, $\boldsymbol{x}$, and a set of new class labels $\Lambda^{[\mathrm{te}]} = \left\{\lambda_1^{[\mathrm{te}]}, \ldots, \lambda_L^{[\mathrm{te}]}\right\}$ with the signature $\mathbb{Y}^{[\mathrm{te}]} = \left\{\boldsymbol{y}_1^{[\mathrm{te}]}, \ldots, \boldsymbol{y}_L^{[\mathrm{te}]}\right\}$ can be stated as

$$\lambda_i^{[\mathrm{te}]} \in \operatorname*{argmax}_i \boldsymbol{x}^{\top} \boldsymbol{W} \boldsymbol{y}_i^{[\mathrm{te}]} \ . \tag{3.40}$$

The solution to the problem (3.39) is based on the following choice of the regularization, which is

$$\Omega(\boldsymbol{W}; \mathbb{Y}, \mathbb{X}) = \alpha \|\boldsymbol{W}\mathbb{Y}\|_{\mathrm{Fro}}^2 + \beta \|\mathbb{X}^{\top}\boldsymbol{W}\|_{\mathrm{Fro}}^2 + \gamma \|\boldsymbol{W}\|_{\mathrm{Fro}}^2. \tag{3.41}$$

In (3.41), $\|\cdot\|_{\mathrm{Fro}}^2$ denotes the Frobenius norm (see appendix) and the scalars α, β, γ are hyper-parameters. The rationale behind (3.41) is as follows. The first term is to ensure that all signatures have a similar Euclidean norm in the instance feature space. This should allow for fair comparisons between the signature vectors $\boldsymbol{y}$ and help in the case of unbalanced training sets. The second term ensures that the variance of the instances features on the attribute space is bounded. This should prevent an over-fitting on the training instances. The third term is there to keep the value range of the weight matrix

$\boldsymbol{W}$ bounded. With the choice of the hyper-parameters $\gamma = \alpha\beta$ and the following loss function

$$\mathcal{L}(\boldsymbol{P}, \boldsymbol{C}) = \|\boldsymbol{P} - \boldsymbol{C}\|^2_{\text{Fro}} \ , \tag{3.42}$$

the solution can be expressed in a closed form as

$$\boldsymbol{W} = \left(\mathbb{X}\mathbb{X}^\top + \alpha \boldsymbol{I}\right)^{-1} \mathbb{X}\boldsymbol{C}\mathbb{Y}^\top \left(\mathbb{Y}\mathbb{Y}^\top + \beta \boldsymbol{I}\right)^{-1} \ , \tag{3.43}$$

where $\boldsymbol{I}$ is the unit matrix, where $\mathbb{X} = \mathbb{X}^{[\text{tr}]}$ and $\mathbb{Y} = \mathbb{Y}^{[\text{tr}]}$ are set for better readability.

3.7.2 Probabilistic Zero-Shot Classification with Semantic Rankings

The core idea of the *PR* approach is the use of pre-trained classifiers in combination with a ranking-based representation of semantic similarity (Hamm and Belkin, 2015). The latter aspect allows for the aggregation of semantic information from multiple heterogeneous sources.

A semantic ranking can be inferred by inspecting the closeness (or similarity as a special case) of an object to other objects or by human judgement. For example, from a set of objects $\Lambda = \{horse, mouse, dog, elephant, fly\}$, the semantic ranking of *elephant* could be stated as $\pi_{elephant} : horse \succ dog \succ mouse \succ fly$, if we think of closeness in terms of size. This concept plays a crucial role in the model that is described next.

The model combines two kinds of top-K probabilistic ranking models and classification approaches. We will focus on the top-K ranking versions of the Mallows (Fligner and Verducci, 1986; Mallows, 1957) and the Plackett–Luce model (Luce, 1959; Marden, 1995; Plackett, 1975), since this particular combination provides good performance in the experimental evaluation in (Hamm and Belkin, 2015). For the classification approach, the multi-class loss is used with multinomial logistic regression. Alternative classification approaches would include one-vs-rest and one-vs-one.

The interrelation of the model parts is as follows. The multinomial logistic regression model is learned on training data, which includes training instances and labels. For a given instance $\boldsymbol{x}$, it is possible with this model to produce the score $f_i(\boldsymbol{x})$, which corresponds to the confidence of predicting a training class label i for the instance $\boldsymbol{x}$. The set of scoring functions $\{f_i\}$, $1 \leq i \leq M$ is used then to define the parameters of the first ranking model over M many training class labels. The second ranking model produces a consensus ranking over multiple semantic rankings associated with a test class

label. The probability of this consensus ranking, together with the scoring functions, will be evaluated with the first model relating to a new test instance vector. The main assumption underlying this approach is that semantic similarity is strongly correlated with the instance features. We will proceed with the formal description of the model.

The probability for the test class $\lambda^{[\mathrm{te}]} \in \Lambda^{[\mathrm{te}]}$ for a given instance vector, $\boldsymbol{x}$, can be stated as

$$\mathbf{P}\left(\lambda^{[\mathrm{te}]} \,|\, \boldsymbol{x}\right) = \sum_{\pi \in \mathbb{S}_M} \mathbf{P}\left(\pi \,|\, \boldsymbol{x}\right) \cdot \mathbf{P}\left(\lambda^{[\mathrm{te}]} \,|\, \pi\right) \ , \tag{3.44}$$

where the second term of the right-hand side is

$$\mathbf{P}\left(\lambda^{[\mathrm{te}]} \,|\, \pi\right) = \frac{\mathbf{P}\left(\lambda^{[\mathrm{te}]}\right) \mathbf{P}\left(\pi \,|\, \lambda^{[\mathrm{te}]}\right)}{\sum\limits_{\lambda \in \Lambda^{[\mathrm{te}]}} \mathbf{P}(\lambda)\mathbf{P}(\pi \,|\, \lambda)} \ . \tag{3.45}$$

The most probable test label is searched for at the prediction phase and this can be evaluated with

$$\hat{\lambda}^{[\mathrm{te}]} \in \operatorname*{argmax}_{\lambda \in \Lambda^{[\mathrm{te}]}} \mathbf{P}\left(\lambda^{[\mathrm{te}]} \,|\, \boldsymbol{x}\right) \ . \tag{3.46}$$

The first ranking model is used to express $\mathbf{P}\left(\pi \,|\, \boldsymbol{x}\right)$ in Equation (3.44), and it is the top-K version of the Plackett–Luce model. It can be stated as

$$\mathbf{P}(\pi \,|\, \boldsymbol{v}) = \prod_{i=1}^{K} \frac{v_{\pi(i)}}{\sum\limits_{j=i}^{M} v_{\pi(j)}} \ , \tag{3.47}$$

where the model parameters are defined as $v_i = \exp(f_i(\boldsymbol{x}))$, where f_i is a scoring function obtained from a multi-class classifier. More details of the Plackett–Luce model will be explained in conjunction with the dyad ranking models in the upcoming chapter.

The second model is used to learn the probability $\mathbf{P}\left(\pi \,|\, \lambda^{[\mathrm{te}]}\right)$ in Equation (3.45). Here the semantic rankings are aggregated to a consensus ranking using the Mallows model (Mallows, 1957), or more concretely the top-K version from Fligner and Verducci (1986). The Mallows model can be considered as a discrete analog of the Gaussian distribution on rankings. It is a distance-based model that belongs to the family of exponential distributions. Its specification is based on a location parameter, $\pi_0 \in \mathbb{S}_K$ (mode, center), and a spread parameter, $\theta \geq 0$:

$$\mathbf{P}(\pi \,|\, \theta, \pi_0) \ = \ \frac{1}{\phi(\theta)} \exp\big(- \theta \cdot D(\pi, \pi_0)\big) \ , \tag{3.48}$$

Algorithm 4 PR Training Algorithm

Require: Input $\mathcal{D}_1 = \{\boldsymbol{x}_n, \lambda_n^{[\text{tr}]}\}_{n=1}^{N_{\text{tr}}}$, $\mathcal{D}_2 = \{\pi_n, \lambda_n^{[\text{te}]}\}_{n=1}^{N_{\text{te}}}$, Parameter K $(1 \leq K \leq M)$

1: **Step 1**
2: Train multi-class classifier using $\mathcal{D}_1$
3: Result: Scoring functions $f_1, \ldots, f_M$
4: **Step 2**
5: Apply Top-K Mallows Model on $\mathcal{D}_2$
6: Result: Consensus ranking π_0^{λ} for each $\lambda \in \Lambda^{[\text{te}]}$
7: **return** results

where D is a distance function on $\mathbb{S}_K$ (typically the Kendall distance) and $\phi(\theta)$ a normalization constant.

After the training, as outlined in Listing 4, predictions can be carried for a new instance vector $\boldsymbol{x}$ using the following maximum a posteriori classifier,

$$\hat{\lambda}^{[\text{te}]} \in \underset{\lambda \in \Lambda^{[\text{te}]}}{\operatorname{argmax}} \prod_{i=1}^{K} \frac{\exp\left(f_{\pi_0^{\lambda}(i)}(\boldsymbol{x})\right)}{\sum_{j=i}^{M} \exp\left(f_{\pi_0^{\lambda}(j)}(\boldsymbol{x})\right)} \ , \tag{3.49}$$

which returns the prediction for the most probable test class label.

4 Generalized Plackett–Luce Models for Dyad Ranking

4.1 Plackett-Luce Model

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

$$\mathbf{P}(\pi \,|\, \boldsymbol{v}) \;=\; \prod_{i=1}^{K} \frac{v_{\pi(i)}}{v_{\pi(i)} + v_{\pi(i+1)} + \ldots + v_{\pi(K)}} = \prod_{i=1}^{K-1} \frac{v_{\pi(i)}}{\sum_{j=i}^{K} v_{\pi(j)}} \;. \tag{4.1}$$

This model is a generalization of the well-known Bradley–Terry model (Bradley and Terry, 1952; Marden, 1995), a model for the pairwise comparison of alternatives which specifies the probability that "a wins against b" in terms of

$$\mathbf{P}(a \succ b) \;=\; \frac{v_a}{v_a + v_b} \;. \tag{4.2}$$

Obviously, the larger v_a in comparison with v_b, the higher the probability that a would be chosen. Likewise, the larger the parameter v_i in (4.1) in comparison with the parameters v_j, $j \neq i$, the higher the probability that y_i would appear in a top rank.

The term "skill" is sometimes used interchangeably for the PL parameter v_i, referring to the fact that in one of the early applications of the model a parameter is associated with a particular horse in a horse race (Marden, 1995).

An example of a distribution represented by the Plackett–Luce model is given in Figure 4.1. There are three options. For each permutation of the three options, a real value from the interval $[0, 1]$ is assigned to it. Since the distribution is discrete, we speak of probability masses that are assigned to the rankings.

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 i-th option in a vase filled with

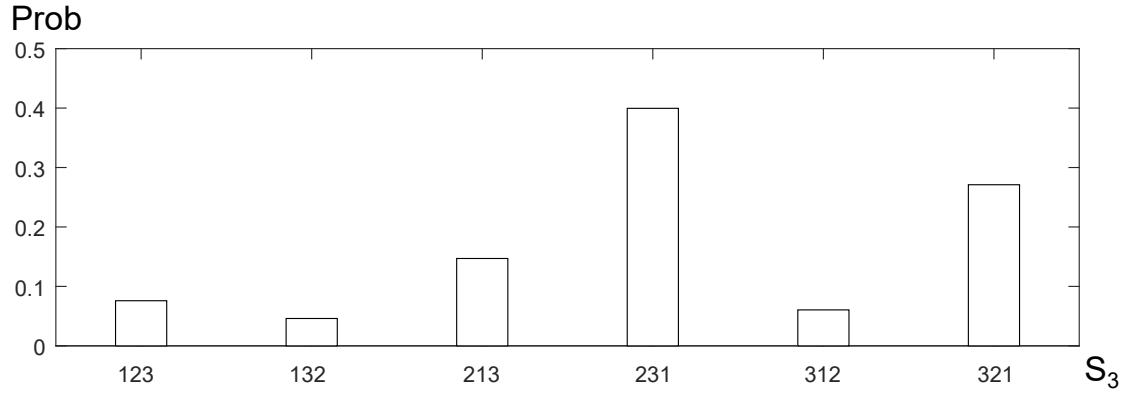

Figure 4.1: The probability distribution of the Plackett-Luce model on rankings of length three.

labeled balls, then $\mathbf{P}(\pi \,|\, \boldsymbol{v})$ is the probability to produce the ranking π by randomly drawing balls from the vase in a sequential way and putting the option drawn in the k-th trial on the position k (unless the option was already chosen before, in which case the trial is annulled). This is why the PL model is also referred to as a "multistage" model.

A nice feature of the Plackett–Luce model is the ability to cope with incomplete rankings. Assume that an incomplete ranking contains J options instead of K, where $J < K$. The model can cope with them in a straightforward way because the probability of an incomplete ranking is given by

$$\mathbf{P}(\pi' \,|\, \boldsymbol{v}) \;=\; \prod_{i=1}^{J} \frac{v_{\pi'(i)}}{v_{\pi'(i)} + v_{\pi'(i+1)} + \ldots + v_{\pi'(J)}} \;\;, \tag{4.3}$$

i.e. by an expression of exactly the same form as (4.1), except that the number of factors is J instead of K. Probability theory provides the same result by "marginalizing out," i.e. summing over all probabilities of π of length K which include extensions of the ranking π' of length J. For example, extensions of the ranking 12 to the ranking of length three would be 123, 132, and 312. The calculation of the marginals is clearly much simpler via (4.3), especially for large K and small J. Marginals are the probabilities of rankings over a subset of the options, and they can be computed easily for this model.

4.1.1 Basic Properties

Parameters The Plackett–Luce parameters are determined only up to a positive multiplicative constant because

$$\mathbf{P}(\pi \mid s \cdot \boldsymbol{v}) = \prod_{i=1}^{K-1} \frac{s \cdot v_{\pi(i)}}{\sum_{j=i}^{K} s \cdot v_{\pi(j)}} = \prod_{i=1}^{K-1} \frac{s \cdot v_{\pi(i)}}{s \cdot \sum_{j=i}^{K} v_{\pi(j)}} = \mathbf{P}(\pi \mid \boldsymbol{v}) \ .$$

To make the model parameters identifiable (or equivalently to choose an equivalence class), one can set one of the latent utilities to zero, e.g. $v_K = 1$. Another possibility is to restrict the sum of the utilities to 1, i.e. to normalize them with $v_i = \frac{v_i}{\sum_j^K v_j}, 1 \leq i \leq K$ (Gormley and Murphy, 2010).

Mode Ranking The ranking with the highest probability of the probability distribution can be obtained in a straightforward way: just by sorting the alternative associated PL utilities $v_1, \ldots, v_K$ in descending order. The probability of this and any other ranking can then be obtained in $\mathcal{O}(K)$ respectively, see Listing 12 in the appendix.

4.1.2 Foundations

4.1.2.1 Luce Choice Axiom

Luce's choice axiom (LCA) is part of a probabilistic choice theory based on two characteristics. First, it is probabilistic, and second, the probability of choosing an alternative from one set is related to the probability of choosing the same alternative from a larger set of alternatives. The key point is that the standard axioms of probability theory alone are not sufficient because they do not offer "a connection between measures over different sets composed of some of the same alternatives" (Pleskac, 2013). LCA establishes this connection, namely the connection on how a decision maker selects an alternative from a larger set is related to when he or she selects the same alternative from a smaller set (and the other way around).

An example to illustrate the axiom is from Guiver and Snelson (2009) and can be stated as follows: Suppose there are four options $\{A, B, C, D\}$, and the corresponding probabilities of choosing from this set are (p_A, p_B, p_C, p_D). Now, if a subset $\{A, C\}$ with choice probabilities (q_A, q_C) is considered, then LCA states that $q_A/q_C = p_A/p_C$, thus meaning that the probability ratio between two options is independent of any other option of the set.

The consequences of the LCA are known in the literature as IIA— the independence from irrelevant alternatives and that choice probability is ratio scale (Luce, 1959). The Plackett–Luce model is the distinct result of the claims being made with the LCA.

4.1.2.2 From the Choice Axiom to the PL Model

Let use define $\mathbf{P}_B(i)$ to be the probability in such a way that i is the preferably chosen option from the set B of several options. Luce stated a ranking postulate, which was later generalized with the concept of L-decomposability for models based on choice probabilities (Critchlow and Fligner, 1993). The postulate states that the probability for every ranking π of length K is

$$\mathbf{P}(\pi) = \mathbf{P}_{\{\pi(1),\ldots,\pi(K)\}}(\pi(1)) \cdot \mathbf{P}_{\{\pi(2),\ldots,\pi(K)\}}(\pi(2)) \cdot \cdots \cdot \mathbf{P}_{\{\pi(K-1),\pi(K)\}}(\pi(K-1)) \quad (4.4)$$

Luce began exploring (4.4) further with choice probabilities adhering to the LCA, which are probabilities of the form

$$\mathbf{P}_B(i) = \frac{p_i}{\sum_{j \in B} p_j} \ . \quad (4.5)$$

The combination of (4.4) and (4.5) indeed yields the PL model stated in Eq. (4.1) (Luce, 1959). Plackett later worked independently on a system of logistic models and his "first order model" coincided with Eq. (4.1) too (Plackett, 1975).

4.1.2.3 Connection to Thurstone's Case V Model

In Thurstone's Case V model, each alternative i has an associated unobserved but continuous random variable, Z_i (Thurstone, 1927).

$$\left(\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(K)\right) \Longleftrightarrow z_{\pi^{-1}(1)} < z_{\pi^{-1}(2)} < \ldots < z_{\pi^{-1}(K)}. \quad (4.6)$$

The probability of observing a particular ordering is equal to the probability of the corresponding order of the random variable realizations. Figure 4.2 shows exemplarily the distribution of three random variables Z_i.

The connection between the model (4.6) and the Plackett–Luce model (4.1) was found by Yellott Jr (1977). Yellott stated that Thurstone's Case V model fulfills Luce's choice axioms only if $-Z_a$ are distributed according to the Gumbel (double exponential) distribution (Guiver and Snelson, 2009; Marden, 1995).

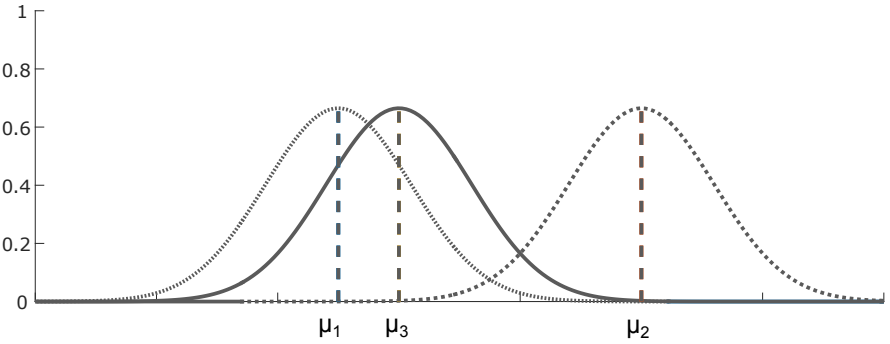

Figure 4.2: Thurstone's Case V model with three normally distributed random variables having equal standard deviations. The corresponding ranking of alternatives would be $2 \succ 3 \succ 1$.

4.2 Joint-Feature Plackett-Luce Model

A generalization of the PL model (4.1) for the dyad ranking setting is introduced next, where feature vectors[1] are used to define the PL parameters $\boldsymbol{v}$ by means of (multivariate) functions. They are modeled as log-linear functions of a dyad with a weight vector as a parameter:

$$v(\boldsymbol{z}) = v(\boldsymbol{x}, \boldsymbol{y}) = \exp\big(\langle \boldsymbol{w}, \Phi(\boldsymbol{x}, \boldsymbol{y}) \rangle\big) \ , \tag{4.7}$$

where $\Phi(\cdot)$ is a joint-feature map as commonly used in structured (output) prediction (Tsochantaridis et al., 2005). The model is termed for this reason as *Joint-Feature Plackett-Luce model* (**JFPL**). The probability for a ranking

$$\rho : \Big(\boldsymbol{x}_{\pi(1)}, \boldsymbol{y}_{\pi(1)}\Big) \succ \Big(\boldsymbol{x}_{\pi(2)}, \boldsymbol{y}_{\pi(2)}\Big) \succ \ldots \succ \Big(\boldsymbol{x}_{\pi(M)}, \boldsymbol{y}_{\pi(M)}\Big) \ ,$$

can thus be stated with (4.7) as

$$\mathbf{P}(\rho \,|\, \boldsymbol{w}) = \mathbf{P}((\varrho, \pi) \,|\, \boldsymbol{w}) = \sum_{m=1}^{M} \frac{\exp\Big[\boldsymbol{w}^\top \Phi\Big(\boldsymbol{x}_{\pi(m)}, \boldsymbol{y}_{\pi(m)}\Big)\Big]}{\sum\limits_{l=m}^{M} \exp\Big[\boldsymbol{w}^\top \Phi\Big(\boldsymbol{x}_{\pi(l)}, \boldsymbol{y}_{\pi(l)}\Big)\Big]} \ . \tag{4.8}$$

A ranking over dyads is considered, because the relation of and the interaction between the objects of a dyad are of importance. This circumstance encompasses two situations. The first is when one object is part of every dyad within a ranking (cf. Section 2.1—a contextual dyad ranking). And the second is when a ranking consists of dyads with

[1] These vectors may alternatively be called covariates, side- or auxiliary information.

differing object pairings. An example for a real-world application for the first situation is the preference of a person for listening to a specific style of music $\boldsymbol{y}$ depending on a specific geographical location $\boldsymbol{x}$. Here, a ranking of songs could be stated in regard to the position of a person. An application for the second situation is similarity learning, where a pair of similar things should be ranked higher than a pair of dissimilar things. Another real-world application is the ability of a driver $\boldsymbol{x}$ to compete against other drivers in regard to a personalized context (car and driving strategy) $\boldsymbol{y}$ for a race. Here, a ranking could be stated that consists of dyads with differing driver and context pairings.

4.2.1 Inference

A ranking of dyads $\varrho = \left\{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_M\right\}$ can be predicted with the given weight vector $\boldsymbol{w}$ by applying (4.7) on every $\boldsymbol{z} \in \varrho$. The dyads can then be ranked according to the obtained parameter values $v(\cdot)$. The resulting ranking corresponds to the above-mentioned *mode* ranking of the PL distribution.

More precisely, a function $h(\varrho)$ is a dyad ranker computed with

$$\hat{\pi} = h(\varrho) = \underset{i=1\ldots M}{\operatorname{argsort}}\ v_i \ ,$$

on a set of latent utilities. As a result, the ranking of dyads within the set ϱ can then be stated as

$$\boldsymbol{z}_{\hat{\pi}(1)} \succ \boldsymbol{z}_{\hat{\pi}(2)} \succ \cdots \succ \boldsymbol{z}_{\hat{\pi}(M)} \ .$$

4.2.2 Geometric Interpretation

A dyad ranking obtained by the aforementioned rule using (4.7) is equivalent to arranging dyads according to

$$\boldsymbol{z}_i \succ \boldsymbol{z}_j \iff \left\langle \boldsymbol{w}, \Phi(\boldsymbol{z}_i) \right\rangle > \left\langle \boldsymbol{w}, \Phi(\boldsymbol{z}_j) \right\rangle \ ,$$

where $\Phi(\boldsymbol{z}) = \Phi(\boldsymbol{x}, \boldsymbol{y})$. A ranking over dyads can be induced for any weight vector $\boldsymbol{w}$, by projecting the joint-feature vectors of their corresponding dyads on to $\boldsymbol{w}$ with $\mathrm{Proj}_{\boldsymbol{w}}(\Phi(\boldsymbol{z}))$ (c.f. appendix). The lengths of the projected vectors indicate the preference strengths and thus determine the ranks of the respective dyads in a ranking. Alternatively, one can use the signed distances of the joint-feature vectors to a hyperplane having $\boldsymbol{w}$ as its normal vector. An example of this is given in Figure 4.3.

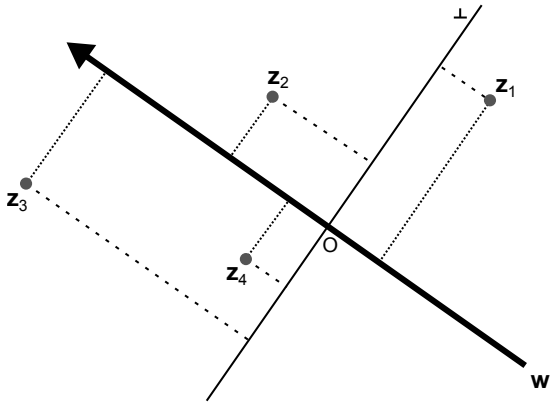

Figure 4.3: Geometric interpretation of the JFPL model. The dyad ranking obtained with the weight $\boldsymbol{w}$ corresponds to $\boldsymbol{z}_3 \succ \boldsymbol{z}_2 \succ \boldsymbol{z}_1 \succ \boldsymbol{z}_4$.

4.2.3 Properties

The JFPL model can be seen as a canonical approach to the dyad ranking setting. It covers all properties that the setting offers: The rankings of different lengths can be modeled as well as the special case of *contextualized* dyad rankings. The JFPL model also puts no restrictions on the domain of the dyad member's feature vectors. What is necessary, however, is a joint-feature vector representation for the dyads.

There are many possibilities for the definition of joint-feature vectors. For example, a joint-feature vector could be made up of the components of a higher-order polynomial, which could, with a selection of some of those features, take on the following form:

$$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{x}_7^3, \boldsymbol{x}_3, \boldsymbol{y}_5^2, \boldsymbol{y}_1, \boldsymbol{x}_4^2 \boldsymbol{y}_2, \boldsymbol{x}_1 \boldsymbol{y}_2, \boldsymbol{x}_2 \boldsymbol{y}_4, 1\right) \quad . \tag{4.9}$$

Another example for the construction of a joint-feature map was already provided by the win–loss tie approach for CR in Chapter 3.4.2. There, the statistical properties about a user-item rating matrix are used to compose a joint-feature vector with 17 joint-features.

The ability of quantifying the certainty (or likewise uncertainty) of a dyad ranking is a further important property. It can be used in various of ways, e.g. for improving visualizations (c.f. Section 5.2.1), for balancing exploration and exploitation (c.f. Section 6.3.1.1), or for speeding up the learning process (c.f. Sections 6.4.3 and 7.5.1). Beyond that, the possibility for a learner to abstain on pairwise preferences is also based on a probabilistic foundation (Cheng et al., 2012). It enables the learner to declare pairwise

preferences as incomparable on which the predictions are uncertain. This scenario is similar the classification with a reject-option in which a classifier may refuse the prediction of a class label. This is useful if the refusal of a class prediction is less costly than to make an unreliable and potentially false prediction. In the case of ranking, there is a further possibility besides a total rejection and full acceptance. It is the possibility of predicting a partial order instead of a total order (ranking).

4.2.4 Connection to Discrete Choice Models

Approaches similar to the JFPL model can be found in the econometrics and psychometrics literature where covariates[2] play a role in explaining (representative) utility. In particular, the *rank ordered logit* or the otherwise known *exploded logit* model is essentially equivalent to the JFPL model regarding the specification of the covariates. These are developed from a different theoretical framework and typically stated without the notion of a joint-feature map $\phi(\boldsymbol{x}, \boldsymbol{y})$.

Logit Model The derivation of the logit model from order statistic (random utility) models follows that of (Train, 2009). A decision maker (or user), n, chooses from among K alternatives. The utility (or profit) of choosing alternative j by the user n is expressed by U_{nj}. This quantity might be known by the user n, but is unknown to an external observer (e.g. a researcher). The behavioral model assumed behind the choice of the user for an alternative i is

$$U_{ni} > U_{nj} \ \ \forall j \neq i \ \ .$$

A researcher is not able to observe U_{ni}, but might know the attributes of the alternatives y_{ni} and the attributes of the user x_n. And the attributes of the alternatives are assumed to be related to the user n. This knowledge enables a researcher to specify a representation of the utility U_{ni} denoted by V_{ni}. And this *representative utility* is usually a function of the attributes $V_{ni} = V(x_n, y_{ni})$ which involves parameters that need to be estimated from data. In random utility models (RUM) it is assumed that $U_{nj} \neq V_{nj}$, i.e. there are aspects that the researcher cannot observe and hence the utility is a composition of

$$U_{ni} = V_{ni} + \epsilon_{ni} \ \ . \tag{4.10}$$

[2] In ML terminology referred to as instances, feature vectors, or attributes.

Table 4.1: The RUM model type and the form of the integral (4.11) depend on the specification of $f(\cdot)$.

Model	Integral has closed-form	Distribution of $f(\cdot)$
logit	yes	(i.i.d) extreme value
nested logit	yes	generalized extreme value
probit	no	multivariate normal
mixed logit	no	any + extreme value

The error ϵ_{ni} captures aspects that affect the utility, but are not included in the representation V_{ni}. It is treated as a random variable, and the vector of errors is defined as $\boldsymbol{\epsilon}_n^\top = (\epsilon_{n1}, \ldots, \epsilon_{nK})$. The probability that the user n chooses an alternative i can be stated with it as

$$\begin{aligned}\mathbf{P}_{ni} &= \mathbf{P}[U_{ni} > U_{nj} \ \ \forall i \neq j] \\ &= \mathbf{P}[V_{ni} + \epsilon_{ni} > V_{nj} + \epsilon_{nj} \ \ \forall i \neq j] \\ &= \mathbf{P}[\epsilon_{nj} - \epsilon_{ni} < V_{ni} - V_{nj} \ \ \forall i \neq j] \ ,\end{aligned}$$

which corresponds to the cumulative distribution, in which each random term $\epsilon_{nj} - \epsilon_{ni}$ is below the quantity $V_{ni} - V_{nj}$. Together with the joint density $f(\boldsymbol{\epsilon}_n)$ of the random error vector, the probability can be expressed as the integral

$$\mathbf{P}_{ni} = \int_\epsilon I[\epsilon_{nj} - \epsilon_{ni} < V_{ni} - V_{nj} \ \ \forall i \neq j] f(\boldsymbol{\epsilon}_n) d\boldsymbol{\epsilon}_n \ . \tag{4.11}$$

The specification of the density $f(\boldsymbol{\epsilon})$, which corresponds to the assumptions about the distribution of the unobservable factors ϵ, determines different kinds of models (see Table 4.1). In the case of the logit model, the ϵ_{nj}'s follow an extreme value distribution, or more precisely, the *type I extreme value* or *Gumbel* distribution. Its cumulative distribution function (CDF) is defined as

$$F(\epsilon_{nj}) = \exp(-\exp(-\epsilon_{nj})) \ , \tag{4.12}$$

and its probability density function (PDF) is given as

$$f(\epsilon_{nj}) = \exp(-\epsilon_{nj}) \exp(-\exp(-\epsilon_{nj})) = \exp(-\epsilon_{nj} - \exp(-\epsilon_{nj})) \ .$$

The full specification of the Gumbel CDF is $F(x|\mu, \beta) = \exp(-\exp(-(x-\mu)/\beta))$ and the PDF is given by $f(x|\mu, \beta) = 1/\beta \exp(-(x-\mu)/\beta - \exp(-(x-\mu)/\beta))$. The

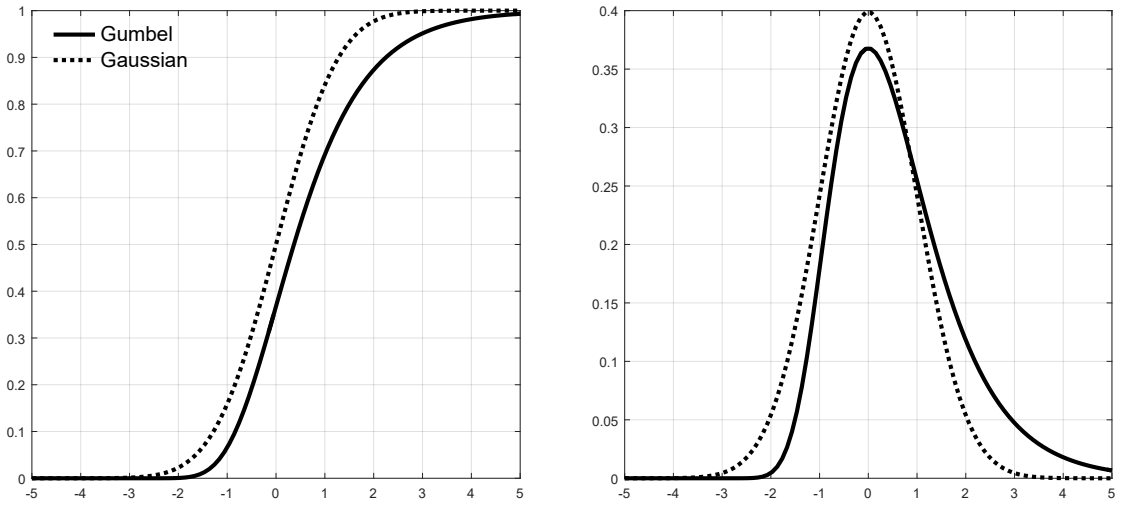

Figure 4.4: Comparison of (standard) Gumbel and (standard) Gaussian distributions. Left: cumulative distribution functions. Right: probability distribution functions.

standard Gumbel refers to the configuration $\mu = 0$, $\beta = 1$, and Figure 4.4 shows it in comparison with the normal (Gaussian) distribution. The PDFs and CDFs of the normal (or Gaussian) distribution are given by $F(x) = \Phi((x - \mu)/\sigma)$ with

$$\Phi(x) = \frac{1}{\sqrt{(2\pi)}} \int_{-\infty}^{x} \exp(-t^2/2)dt$$

and $f(x|\mu, \sigma^2) = (\sqrt{2\sigma^2\pi})^{-1} \exp(-(x - \mu)^2/(2\sigma^2))$ respectively. The standard normal distribution refers to the configuration in which there is zero mean and unit variance ($\mu = 0, \sigma^2 = 1$). From the figure it can be seen that the Gumbel PDF is not symmetric, and it is right-skewed in comparison with the normal PDF (fatter right tail). The difference between the errors that are distributed Gumbel or Gaussian is barely distinguishable empirically (Train, 2009).

The difference between each two extreme value-distributed variables ϵ_{nj} and ϵ_{ni}, i.e. $\bar{\epsilon}_{nji} = \epsilon_{nj} - \epsilon_{ni}$, follows the logistic distribution

$$F(\bar{\epsilon}_{nji}) = \frac{\exp(\bar{\epsilon}_{nji})}{1 + \exp(\bar{\epsilon}_{nji})} \quad .$$

The key property of the logit is the underlying assumption that the unobserved portions of utility (the ϵ_{nj}'s) are independent—this gives rise to criticism and motivation for alternative models. Other discrete choice models, such as the mixed logit, provide more

flexibility in that the errors can be correlated. However, the rather strong assumption underlying the logit can also be seen as a positive characteristic. This is because if the representative utility V_{nj} is modeled well, then the remaining aspects, i.e. the unobservable portion of utility ϵ_{nj}, should only contain white noise. From this perspective, the logit *just* requires a *correct* representation $V(\cdot, \cdot)$.

If correlations between alternatives are of concern in logits, then they must be modeled explicitly so that the errors remain independent[3]. With the choice of the cumulative distribution (4.12) and the independence of the ϵ's, we can rewrite the choice probability (4.11) as

$$\mathbf{P}_{ni} = \int \Big(\prod_{i \neq j} \exp(-\exp(-(\epsilon_{ni} + V_{ni} - V_{nj}))) \Big) \exp(-\epsilon_{ni}) \exp(-\exp(-\epsilon_{ni})) d\epsilon_{ni} \ . \quad (4.13)$$

The integral (4.13) has a closed-form expression, which is the logit choice probability (McFadden, 1973):

$$\mathbf{P}_{ni} = \frac{\exp(V_{ni})}{\sum_j \exp(V_{nj})} \ , \quad (4.14)$$

where the representative utility V_{nj} is often specified as the function $V(\boldsymbol{x}^{(nj)}) = \boldsymbol{\beta}^\top \boldsymbol{x}^{(nj)}$, which is linear in its parameter vector $\boldsymbol{\beta}$. The algebraic operations to get from (4.13) to (4.14) are provided in the appendix. The possibility to express probabilities in a closed-form is the main advantage of the logit—this also makes it different to the other models.

Rank Ordered Logit Model (ROL) An extension of the logit for rank-ordered observations is reported in (Allison and Christakis, 1994; Luce and Suppes, 1965); it relies on the theoretical work of Marschak (1959).

It proceeds from the observation that the user i made a ranking of the options A, B, C and D. And the utility of an option can be specified as $U_{ij} = \boldsymbol{\beta}^\top \boldsymbol{x}^{(ij)} + \epsilon_{ij}$ with $j = A, B, C, D$, where ϵ_{ij} is following the Gumbel distribution. Then, the probability of observing a particular ranking of the alternatives can be expressed as the product of multiple standard logit models as follows:

$$\Pr(B \succ D \succ A \succ C) = \frac{\exp(\boldsymbol{\beta}^\top \boldsymbol{x}^{(iB)})}{\sum_{j=A,B,C,D} \exp(\boldsymbol{\beta}^\top \boldsymbol{x}^{(ij)})} \frac{\exp(\boldsymbol{\beta}^\top \boldsymbol{x}^{(iD)})}{\sum_{j=A,C,D} \exp(\boldsymbol{\beta}^\top \boldsymbol{x}^{(ij)})} \frac{\exp(\boldsymbol{\beta}^\top \boldsymbol{x}^{(iA)})}{\sum_{j=A,C} \exp(\boldsymbol{\beta}^\top \boldsymbol{x}^{(ij)})} \ . \quad (4.15)$$

[3] At least if they are used for predictive purposes.

Each logit in (4.15) refers to a *pseudo-observation* in which an alternative is chosen from a set of remaining alternatives. The sets within consecutive logits are sets that contain all alternatives excluding those that have been chosen before.

4.3 Bilinear Plackett–Luce Model

A particular choice of the joint-feature map is the Kronecker (tensor) product between two vectors; this is specified by

$$\Phi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \otimes \boldsymbol{y} = \big(x_1 \cdot y_1, x_1 \cdot y_2, \ldots, x_r \cdot y_c\big) = \mathbf{vec}\big(\boldsymbol{x}\boldsymbol{y}^\top\big) \ , \tag{4.16}$$

which is a vector of the length $p = r \cdot c$ comprising 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 $\mathbf{W} = (w_{i,j})$, in which an entry $w_{i,j}$ can be considered as the weight of the interaction term $\boldsymbol{x}_i \boldsymbol{y}_j$. This choice of the joint-feature map yields a bilinear version of the PL model:

$$v(\boldsymbol{z}) = v(\boldsymbol{x}, \boldsymbol{y}) = \exp(\langle \boldsymbol{w}, \boldsymbol{x} \otimes \boldsymbol{y} \rangle) = \exp\big(\boldsymbol{x}^\top \mathbf{W} \boldsymbol{y}\big) \ . \tag{4.17}$$

Given the ranking

$$\rho : \Big(\boldsymbol{x}_{\pi(1)}, \boldsymbol{y}_{\pi(1)}\Big) \succ \Big(\boldsymbol{x}_{\pi(2)}, \boldsymbol{y}_{\pi(2)}\Big) \succ \ldots \succ \Big(\boldsymbol{x}_{\pi(M)}, \boldsymbol{y}_{\pi(M)}\Big) \ ,$$

the probability of it can be stated with (4.17) as

$$\mathbf{P}(\rho \,|\, \boldsymbol{W}) = \sum_{m=1}^{M} \frac{\exp\Big[\boldsymbol{x}_{\pi(m)}^\top \boldsymbol{W} \boldsymbol{y}_{\pi(m)}\Big]}{\sum\limits_{l=m}^{M} \exp\Big[\boldsymbol{x}_{\pi(l)}^\top \boldsymbol{W} \boldsymbol{y}_{\pi(l)}\Big]} \ . \tag{4.18}$$

4.3.1 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 interesting question concerns the identifiability of this model. Recall that, for a parameterized class of models $\mathcal{M}$, identifiability requires a bijective relationship between the models $M_\theta \in \mathcal{M}$ and the parameters θ, that is, models are uniquely identified by their parameters. Or, stated differently, the 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 (4.1) with constant skill parameters $\boldsymbol{v} = (v_1, \ldots, v_K)$ is not identifiable, since the model is invariant against the 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. $\mathbf{P}(\pi \,|\, \boldsymbol{v}) = \mathbf{P}(\pi \,|\, \boldsymbol{v}^*)$ for all rankings π. The PL model is, however, 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.

Proposition 1. *Suppose that none of the $\boldsymbol{y}$ features vectors includes a constant feature, i.e. $|\mathbb{Y}_i| > 1$ for each of the domains in (2.3), and that the $\boldsymbol{x}$ feature vectors includes at most one such feature (accounting for a bias) in at least one ranking ρ. Then, the bilinear PL model with skill values defined according to (4.17) is identifiable.*

Proof: Recall that the standard PL model is invariant against multiplication with a positive constant, and that this is the only invariance of the model (c.f. Section 4.1.1). Since the bilinear PL model defined by (4.17) is log-linear in $\mathbf{W}$, invariance on the level of this parameter can only be additive. In the case of the bilinear PL model, this means that the probability distribution over rankings becomes equal if

$$\boldsymbol{x}^\top \mathbf{W} \boldsymbol{y} = \boldsymbol{x}^\top \mathbf{W}^* \boldsymbol{y} + \gamma \ , \tag{4.19}$$

for all dyads in the rankings.

We will now proceed with the contraposition of the model parameter identification condition, which is, $\boldsymbol{W} \neq \boldsymbol{W}^* \Rightarrow \mathbf{P}(\rho \,|\, \boldsymbol{W}) \neq \mathbf{P}(\rho \,|\, \boldsymbol{W}^*)$. To prove this, we will show that the equality $\mathbf{P}(\rho \,|\, \boldsymbol{W}) = \mathbf{P}(\rho \,|\, \boldsymbol{W}^*)$ cannot be obtained.

By denoting the elements of $\mathbf{W}$ and $\mathbf{W}^*$ by $w_{i,j}$ and $w^*_{i,j}$, respectively, (4.19) means that

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

In the first case, suppose that a ranking ρ has at least two dyads $(\boldsymbol{x}_k, \boldsymbol{y}_m)$ and $(\boldsymbol{x}_l, \boldsymbol{y}_n)$, where either the $\boldsymbol{x}$ vectors differ in the feature i or the $\boldsymbol{y}$ vectors differ in the feature j

and where $\Delta w_{i,j}$ does not vanish. Then, it is clear that the γ is not independent of the dyads. In fact, we may evaluate up to four different values of γ which are: γ_{km}, γ_{kn}, γ_{lm}, and γ_{ln}, where γ_{ab} is defined as $\gamma_{ab} = \sum_i \sum_j \Delta w_{i,j} x_{ai} y_{bj}$. Thus, with a varying parameter, $\mathbf{W}$, one obtains different γ values in dependence of the dyads. This, in turn, results in different probability distributions.

The second case is where only one dyad member is changing while the other is repeated or fixed. This corresponds to the situation of a contextual dyad ranking (2.6). Without loss of generality, suppose the $\boldsymbol{x}$ vectors are repeated and there are at least two $\boldsymbol{y}$ vectors, $\boldsymbol{y}_m$ and $\boldsymbol{y}_n$, that differ in the feature j and $\Delta w_{i,j}$ does not vanish for the feature $\boldsymbol{x}_i \neq 0$. The vectors $\boldsymbol{y}_m$ and $\boldsymbol{y}_n$ then have an impact on γ, and a change of the weight parameter $\mathbf{W}$ lead to a change of the probability distribution. Under the stated conditions, the bilinear PL model is identifiable in both cases. □

4.3.2 Connections to Related Models

Comparison between the Linear and Bilinear PL Models It is not difficult to see that the linear PL model (3.5) from Section 3.1.3, subsequently referred to as LinPL, is indeed a special case of the bilinear model (4.17), called **BilinPL**. 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 the 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 (3.5).

The other way around, LinPL can also be applied in the setting of dyad ranking, provided the domain $\mathbb{Y}$ 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 (2.3) and apply a standard label ranking method to the set of these meta-labels. Therefore, both approaches are in principle equally expressive. Still, an obvious problem of this transformation is the potential size[4]

$$K = |\mathbb{Y}| = |\mathbb{Y}_1| \times |\mathbb{Y}_2| \times \ldots \times |\mathbb{Y}_c|$$

of the label set thus produced, which might be huge. In fact, the number of parameters that need to be learned for the model (3.5) is $r \cdot |\mathbb{Y}|$, i.e. $r \cdot a^c$ under the assumption that each feature has a values. For comparison, the number of parameters is only $r \cdot c$ in the

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

bilinear model. Moreover, all information about relationships between the options (such as shared features or similarities) is 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 that the assumptions underlying the bilinear model (4.17) 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 (4.17) is in fact a misspecification: If the features are not meaningful, or if the bilinear model is not properly reflecting their interaction, then learning on the basis of (4.17) cannot be successful.

In this regard, it is also interesting to mention that both approaches can be combined. To this end, the feature vectors $\boldsymbol{y}$ are extended by a (1-of-K) dummy-encoding, i.e. dyad ranking is used with feature vectors of the following form:

$$\boldsymbol{y} = \big(y_1, y_2, \ldots, y_c, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{\text{length } K} \big) \tag{4.20}$$

Using this representation, subsequently called *LinSidePL*, the learner is in principle free to exploit the side-information y_i or to ignore it and only use the dummy labels.

In comparisons with the ROLs from Section 4.2.4, it can be stated that LinPL shares similarities with those ROLs whose factors consist of multinomial logits, whereas BilinPL resembles the case where the factors are conditional logits (Hoffman and Duncan, 1988).

Connection to Logistic Regression It is possible with dyad ranking to mimic label ranking by expressing the $\boldsymbol{y}$ vectors in terms of labels. Moreover, it is possible to draw a link between the BilinPL model and the logistic regression model. In the case where rankings are of length 2, the BilinPL model can resemble the *binary logistic regression* model because the $\boldsymbol{x}$ vectors can be considered as instances and the 1-of-K encoded $\boldsymbol{y}$ vectors as binary classes. This means that the class labels take on the following form: $\boldsymbol{y}_1 = (1, 0)^\top$ and $\boldsymbol{y}_2 = (0, 1)^\top$.

A connection can also be drawn between the LinPL model for label ranking and the *multinomial logistic regression* model, when only the first stage of the PL model is considered (Cheng and Hüllermeier, 2012). This connection can also be established with the BilinPL model (4.17), where again it is required to express the class labels in terms of the 1-of-K encoding. With $\boldsymbol{y}_m := (0, \ldots, 1 \ldots, 0)^\top$ for $1 \leq m \leq M$, the probability of observing the class $c \in \mathcal{Y}$ can be expressed as:

$$\mathbf{P}(c \,|\, \boldsymbol{x}) = \frac{\exp\left(\boldsymbol{x}\boldsymbol{W}\boldsymbol{y}_c\right)}{\sum_{m=1}^{M} \exp\left(\boldsymbol{x}\boldsymbol{W}\boldsymbol{y}_m\right)} = \frac{\exp\left(\langle \boldsymbol{\beta}_c, \boldsymbol{x} \rangle\right)}{\sum_{m=1}^{M} \exp\left(\langle \boldsymbol{\beta}_m, \boldsymbol{x} \rangle\right)} \ , \tag{4.21}$$

where $\boldsymbol{\beta}_j := \boldsymbol{W}\boldsymbol{y}_j$ and the vector $\boldsymbol{x}$ is defined with an intercept term as $\boldsymbol{x} := [\boldsymbol{x}', 1]$, so that an additional bias term α_j is modeled like it is common in multinomial logistic regression. The right-hand side of (4.21) corresponds to the standard notation of a multinomial logistic regression model (Agresti, 2003).

Connection to the Latent Feature Log-Linear Model Recall that the LFL model for dyadic prediction from Section 3.6.1 is capable of modeling the rating preferences of users for items. It is furthermore capable of including side-information which results in the following model,

$$\mathbf{P}(y \,|\, \boldsymbol{x} = (r, c), \theta) \propto \exp\left(\sum_{k=1}^{K} u_{rk}^{(y)} v_{ck}^{(y)} + (\boldsymbol{w}^{(y)})^\top \bar{\boldsymbol{s}}_{rc}\right) \ , \tag{4.22}$$

where $\theta = \{\boldsymbol{U}^{(1)}, \ldots, \boldsymbol{V}^{(R)}, \boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(R)}\}$ and $\bar{\boldsymbol{s}}_{rc} := [\boldsymbol{s}_r, \ \boldsymbol{s}_c, \ \boldsymbol{s}_{rc}]$. This model also exhibits a bilinear structure as well as the BilinPL model (4.17), i.e.

$$\log v(\boldsymbol{z}) = f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\top \boldsymbol{W} \boldsymbol{y} \ .$$

We have seen previously that with a 1-of-K encoding of $\boldsymbol{y}$ vectors, it is possible to represent class labels. In a similar way, user and item entities could be represented as class labels too. By setting $\boldsymbol{x}_i = \boldsymbol{e}_i$ and $\boldsymbol{y}_j = \boldsymbol{e}_j$, it is possible to restate (4.22) as

$$f(\boldsymbol{x}_r, \boldsymbol{y}_c) = \boldsymbol{e}_r^\top \boldsymbol{W} \boldsymbol{e}_c \ . \tag{4.23}$$

This specification, however, results in a model which would not be usable for predictive purposes, since values associated with the combinations (r, c) would rather be "remembered" (Menon, 2013). This problem is tackled with the LFL model by a factorization of $\boldsymbol{W}$. This corresponds to the first summand under the exponential in (4.22).

A common internal bilinear scoring function of both models can be pointed out by considering side-information. To this end, let $\boldsymbol{x}_i = [\boldsymbol{e}_i; \boldsymbol{s}_i]$ and $\boldsymbol{y}_j = [\boldsymbol{e}_j; \boldsymbol{s}_j]$, where $\boldsymbol{e}_i$ identifies the i-th entity and $\boldsymbol{s}_i$ is the side-information for that entity. It is, then, possible to formulate a bilinear scoring function that contains multiple parts:

$$\begin{aligned} f(\boldsymbol{x}_i, \boldsymbol{y}_j) &= \boldsymbol{x}_i^\top \boldsymbol{W} \boldsymbol{y}_j \\ &= \boldsymbol{e}_i^\top \boldsymbol{W}_1 \boldsymbol{e}_j + \boldsymbol{s}_i^\top \boldsymbol{W}_2 \boldsymbol{e}_j + \boldsymbol{e}_i^\top \boldsymbol{W}_3 \boldsymbol{s}_j + \boldsymbol{s}_i^\top \boldsymbol{W}_4 \boldsymbol{s}_j \ , \end{aligned} \tag{4.24}$$

where the model weight $\boldsymbol{W}$ exhibits a block structure, which is illustrated in Figure 4.5.

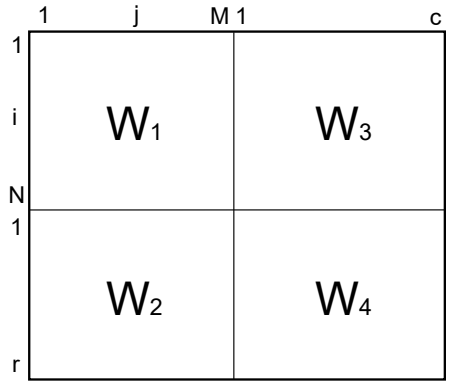

Figure 4.5: Building blocks of the matrix $\boldsymbol{W} = [\boldsymbol{W}_1, \boldsymbol{W}_3; \boldsymbol{W}_2, \boldsymbol{W}_4]$.

The formulation (4.24) exhibits multiple parts. For instance, term two, $\boldsymbol{s}_i^\top \boldsymbol{W}_2 \boldsymbol{e}_j$, when embedded in a PL model, resembles the LinPL model used for label ranking from Section 4.3.2. A model using term four, $\boldsymbol{s}_i^\top \boldsymbol{W}_4 \boldsymbol{s}_j$, would be the most versatile with regard to cold-start predictions. The total bilinear scoring function resembles that of model (4.22) up to the point that the matrix block $\boldsymbol{W}_1$ is not factorized.

To recap, the inclusion of side-information is considered as an interesting option for the LFL base model, and it is applicable if information on users and items is available. For the BilinPL model, the existence of (predictive) feature representation for the domains is a prerequisite. This along with the fact that the LFL model deals with nominal (ordinal) rating data whereas the BilinPL model acts on ranking data can be termed as vital differences between the models. Therefore, an option for future research would be an extension of the BilinPL model for learning latent features.

4.3.3 Learning the Bilinear PL Model

4.3.3.1 Learning via Maximum Likelihood Estimation

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

$$\rho_n : \left(\boldsymbol{x}_1^{(n)}, \boldsymbol{y}_1^{(n)}\right) \succ \left(\boldsymbol{x}_2^{(n)}, \boldsymbol{y}_2^{(n)}\right) \succ \ldots \succ \left(\boldsymbol{x}_{M_n}^{(n)}, \boldsymbol{y}_{M_n}^{(n)}\right) \quad . \tag{4.25}$$

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

$$\mathcal{L}(\boldsymbol{w}; \mathcal{D}) = \mathbf{P}(\mathcal{D} \,|\, \boldsymbol{w}) = \prod_{n=1}^{N} \prod_{m=1}^{M_n - 1} \frac{\exp\left(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_m^{(n)}\right)}{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l^{(n)}\right)} \quad ,$$

where we set $\bar{\boldsymbol{z}}_m^{(n)} := \Phi(\boldsymbol{z}_m^{(n)})$ and where $\Phi(\boldsymbol{z}_m^{(n)}) = \boldsymbol{x}_m^{(n)} \otimes \boldsymbol{y}_m^{(n)}$. Instead of using the formula directly, which involves many product terms, it is more common in MLE to resort to the *log*-likelihood for numerical stability. The reason for this is that the limited floating point representation in common computer architectures can exceed (underflow) if a larger number of small probabilities are multiplied.

As in the case of the linear PL model, the learning problem can now be formalized as finding the maximum likelihood estimate, i.e. the parameter

$$\boldsymbol{w}^* = \operatorname*{argmax}_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}; \mathcal{D}) \quad , \tag{4.26}$$

that maximizes the likelihood (Aldrich et al., 1997; Fisher, 1922) or, equivalently, minimizes the negative log-likelihood (NLL)

$$\ell(\boldsymbol{w}; \mathcal{D}) = -\sum_{n=1}^{N} \sum_{m=1}^{M_n - 1} \boldsymbol{w}^\top \bar{\boldsymbol{z}}_m^{(n)} + \sum_{n=1}^{N} \sum_{m=1}^{M_n - 1} \log\left(\sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l^{(n)}\right)\right) \quad . \tag{4.27}$$

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

$$\frac{\partial \ell}{\partial w_i} = \sum_{n=1}^{N} \sum_{m=1}^{M_n - 1} g(\boldsymbol{w})^{-1} h(\boldsymbol{w}) - \sum_{n=1}^{N} \sum_{m=1}^{M_n - 1} \bar{z}_{m,i}^{(n)}, \tag{4.28}$$

where

$$g_m(\boldsymbol{w}) = \sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l^{(n)}\right) \quad \text{and}$$

$$h_m(\boldsymbol{w}) = \sum_{l=m}^{M_n} \bar{z}_{l,i}^{(n)} \exp\left(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l^{(n)}\right) \quad .$$

For such methods, the convexity of the function to be minimized is of critical importance (Boyd and Vandenberghe, 2004). It is, therefore, shown in the upcoming section that the negative log-likelihood function (4.27) is indeed a convex function.

4.3.3.2 Convexity of the NLL

To show the convexity of the negative log-likelihood function (4.27), recall that

$$\ell(\boldsymbol{w}; \mathcal{D}) = \sum_{n=1}^{N} -\log \mathbf{P}(\rho_n \,|\, \boldsymbol{w}) \tag{4.29}$$

with

$$-\log \mathbf{P}(\rho \,|\, \boldsymbol{w}) = \sum_{m=1}^{M-1} \underbrace{-\boldsymbol{w}^\top \bar{\boldsymbol{z}}_m}_{\text{term 1}} + \sum_{m=1}^{M-1} \underbrace{\log\left(\sum_{l=m}^{M} \exp(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l)\right)}_{\text{term 2}} \,, \tag{4.30}$$

where the index n is dropped in ρ_n and in $\bar{\boldsymbol{z}}_l^{(n)}$ for better readability. Using the fact (e.g. from Luenberger (1973)) that a function comprising a sum of convex functions is convex again, it needs to be analyzed if each of the summands in (4.30) is convex to state the convexity for $\ell(\mathcal{D}, \boldsymbol{w})$. Term 1 is a linear function, which is both convex and concave (in a non-strict sense) at the same time. Term 2 refers to the function

$$f(\boldsymbol{w}) := \log\left(\sum_{l} \exp(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l)\right) \,, \tag{4.31}$$

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

$$\begin{aligned} g(t) := f(\boldsymbol{w} + t\boldsymbol{v}) &= \log\left(\sum_{l} \exp((\boldsymbol{w} + t\boldsymbol{v})^\top \bar{\boldsymbol{z}}_l)\right) \\ &= \log\left(\sum_{l} \exp(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l + t\boldsymbol{v}^\top \bar{\boldsymbol{z}}_l)\right) \,, \end{aligned} \tag{4.32}$$

where $\boldsymbol{v}$ is a vector of the same dimensionality as $\boldsymbol{w}$ and $t \in \mathbb{R}$. The function $g : \mathbb{R} \to \mathbb{R}$ represents a slice of the function f along a line in the direction of $\boldsymbol{v}$. The function f is convex if every such restriction is convex. A criterion for $g(\cdot)$ to be convex is that its second derivative should always be positive. The first derivative of (4.32) is given by

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

with

$$h(t) = \sum_l \exp(\boldsymbol{w}\bar{\boldsymbol{z}}_l + t\boldsymbol{v}^\top \bar{\boldsymbol{z}}_l) \ .$$

The second derivative can be stated then as follows:

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

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

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

The last argument shows that all possible second derivatives are positive because each involved factor is equal or larger than zero, irrespective of a particular choice of $\boldsymbol{v}$. The function f is thus convex for every possible line restriction. And consequently, the NLL (4.29) is a convex function.

4.3.3.3 Convex Optimization Approaches

From an optimization point of view, the NLL (4.27) is a scalar function $\ell : \mathbb{R}^n \to \mathbb{R}$, and the minimization thereof is an unconstrained problem (Luenberger, 1973). Note that in the optimization literature, the function to be minimized is usually denoted by f and the variable as x. To prevent the cluttering of notation, we will continue with the variable names introduced with the BilinPL model. The learning of its parameters will subsequently be expressed in terms of convex optimization. Furthermore, the following abbreviation will be used: The gradient of ℓ is a column vector $g(\boldsymbol{w}) = \nabla\ell(\boldsymbol{w})^\top = (\frac{\partial \ell}{\partial w_1}, \ldots, \frac{\partial \ell}{\partial w_n})^\top$.

Gradient Descent Gradient descent is a simple iterative optimization method for finding a minimum of a function. It is a so-called first-order method because it utilizes the first derivative for finding the direction toward a minimum. The method is specified by the following iterative procedure:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k g(\boldsymbol{w_k}) \ , \tag{4.33}$$

in which α_k is a step size α_k that is allowed to change at every iteration (Boyd and Vandenberghe, 2004). Either one uses a fixed α, which may lead to poor convergence if its value is taken too small. Or otherwise, one selects α at every iteration. The determination of α_k is called *line search*, in which an α_k is searched so that

$$\alpha_k = \underset{\alpha \in \mathbb{R}_{>0}}{\operatorname{argmin}} = \ell(\boldsymbol{w}_k - \alpha g(\boldsymbol{w_k})) \ . \tag{4.34}$$

The line search can either be accomplished exactly (e.g. via the conjugate gradient method) or approximately (e.g. via backtracking).

Newton's Method In Newton's method (also known as the Newton–Raphson method), the idea is based on the approximation of ℓ locally with a quadratic function. To this end, ℓ is approximated near the point $\boldsymbol{w}_k$ using the truncated Taylor series:

$$\ell(\boldsymbol{w}) \simeq \ell(\boldsymbol{w}_k) + g(\boldsymbol{w}_k)(\boldsymbol{w} - \boldsymbol{w}_k) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_k)^\top H(\boldsymbol{w}_k)(\boldsymbol{w} - \boldsymbol{w}_k) \ .$$

The minimizer of the right-hand side leads to the following iterative scheme, which is given by

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \frac{g(\boldsymbol{w}_k)}{H(\boldsymbol{w}_k)} \ . \tag{4.35}$$

We assume that the Hessian matrix H is positive definite at $\boldsymbol{w}^*$, which corresponds to the (second-order) sufficiency criterion for a minimum at $\boldsymbol{w}^*$. The method is well defined near the solution $\boldsymbol{w}^*$ as long as ℓ has continuous partial derivatives.

L-BFGS The limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method is the method of choice for BilinPL. It is a state-of-the-art (unconstrained convex) optimization method that approximates the BFGS algorithm using a limited amount of memory. The method can deal with a large number of variables (high dimensionality of $\boldsymbol{w}$) due to its linear memory requirement by storing only a few vectors for the approximation (Liu and Nocedal, 1989).

4.3.3.4 Optimization via Iterative Majorization

The MM algorithm, proposed by Hunter (2004), belongs to the standard approaches for the maximum likelihood estimation of the basic PL model parameters $\boldsymbol{v}$. The acronym MM stands for majorization–minimization (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 used 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 every 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 the Markov chain. This enables a fast and efficient spectral inference algorithm.

Recall that in contrast to the basic PL model (4.1), the parameters v_i are not real numbers in the BilinPL model (4.17), but the 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 (4.27).

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. $g(\boldsymbol{u}, \boldsymbol{u}) = f(\boldsymbol{u})$, see Figure 4.6 for an example. Moreover, at this point, it should also hold that $g(\boldsymbol{w}, \boldsymbol{u}) \geq f(\boldsymbol{w})$ 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):

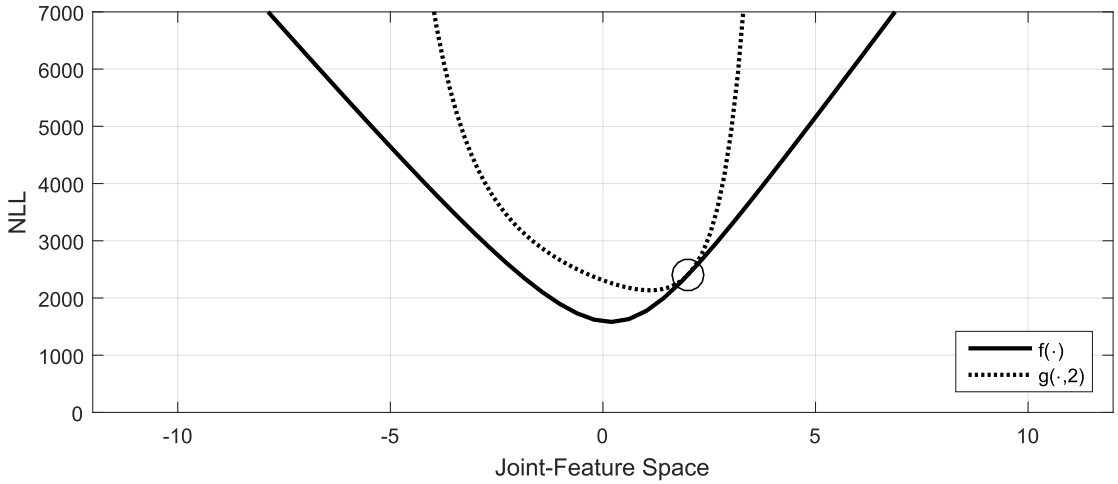

Figure 4.6: Example of a function g (dotted line) that majorizes the BilinPL NLL function f (solid line). The function value of g is equal to that of f at the supporting point $u = 2$ (circle).

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 $f(\boldsymbol{w}) = \ell(\mathcal{D}, \boldsymbol{w})$. To get rid of the logarithm in (4.27) and to simplify the objective, one can exploit the concavity of the logarithm, i.e.

$$\ln(x) \leq \ln(y) + \frac{x}{y} - 1 \ , \tag{4.36}$$

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

$$f(y) \leq f(x) + f'(x)(y - x) \ , \tag{4.37}$$

where the right-hand side of (4.37) is the first-order Taylor approximation of f at the

point x. With (4.36), it is possible to express the surrogate function $g(\boldsymbol{w}, \boldsymbol{u})$ as follows:

$$\begin{aligned} g(\boldsymbol{w}, \boldsymbol{u}) = &\sum_{n=1}^{N} \sum_{m=1}^{M_n} \left(\frac{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l^{(n)}\right)}{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{u}^\top \bar{\boldsymbol{z}}_l^{(n)}\right)} \right) \\ &+ \sum_{n=1}^{N} \sum_{m=1}^{M_n} \log \left(\sum_{l=m}^{M_n} \exp\left(\boldsymbol{u}^\top \bar{\boldsymbol{z}}_l^{(n)}\right) \right) \\ &- \sum_{n=1}^{N} \sum_{m=1}^{M_n} 1 - \sum_{n=1}^{N} \sum_{m=1}^{M_n} \boldsymbol{w}^\top \bar{\boldsymbol{z}}_m^{(n)} \ . \end{aligned} \tag{4.38}$$

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

$$\frac{\partial g}{\partial w_i} = \sum_{n=1}^{N} \sum_{m=1}^{M_n} \left(\frac{\sum_{l=m}^{M_n} \bar{z}_{l,i}^{(n)} \exp\left(\boldsymbol{w}^\top \bar{\boldsymbol{z}}_l^{(n)}\right)}{\sum_{l=m}^{M_n} \exp\left(\boldsymbol{u}^\top \bar{\boldsymbol{z}}_l^{(n)}\right)} \right) - \sum_{n=1}^{N} \sum_{m=1}^{M_n} \bar{\boldsymbol{z}}_m^{(n)} \ , \tag{4.39}$$

$1 \leq i \leq d$. To realize Step 2 of the MM algorithm, i.e. the update step, one needs to find a vector, $\boldsymbol{w}^*$, in such a way 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

$$\boldsymbol{w} = \boldsymbol{u} - [Hg(\boldsymbol{u}, \boldsymbol{u})]^{-1} \nabla f(\boldsymbol{u}) \ . \tag{4.40}$$

The Hessian of g is computationally less complex than that of f. Yet, for the minimization of (4.27), 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 Chapter 7.

4.3.3.5 Online Learning using Stochastic Gradient Descent

An interesting alternative approach, especially in the large-scale learning regime, is stochastic gradient descent (SGD) (Bottou, 1998, 2010). The core idea of SGD is to update model parameters iteratively on the basis of randomly picked examples. The algorithm is explicated for the case of dyad ranking with the BilinPL model in Listing 5. 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 skip parameter updates on examples on which the ranker is certain about which results in a shorter training phase. It is also possible, thanks to the probabilistic information, to implement some kind of active learning via uncertainty sampling (Lewis and Gale, 1994; Settles, 2010). This refers to the technique in which a classifier is applied on unlabeled examples and where a teacher is used to classify a subset of the examples on which the classifier is highly uncertain about.

Algorithm 5 BilinPL with SGD

Require: data set $\mathcal{S}$, initial learning rate η_0, regularization parameter λ, number of 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** $\boldsymbol{w}$

4.3.4 Prediction of Dyad Rankings with the Bilinear PL Model

After obtaining $\boldsymbol{w}$ via (4.26), the resulting model matrix $\boldsymbol{W} = \mathbf{vec}^{-1}(\boldsymbol{w})$ can be used to predict rankings on dyads (for the definition of $\mathbf{vec}^{-1}$, refer to the appendix). To rank the set $\varrho = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_M\}$ of dyads one has to calculate the skills $v(\cdot)$ according to (4.17) and rank the dyads correspondingly similar to the procedure described in Section 4.2.1. That approach produces a ranking of dyads which exhibits the highest probability mass, i.e. the mode of the probability distribution. The probabilistic information that comes along at the inference can also be used, for example, for the prediction of partial orders by abstaining on pairs for which $\mathbf{P}(\boldsymbol{z_i} \succ \boldsymbol{z_j}) \approx \frac{1}{2}$ (Cheng et al., 2012).

The learned weights are furthermore amenable to a geometrical interpretation. To this end, let $\boldsymbol{w}'$ denote a projection into the others' member space, i.e. $\boldsymbol{w}' = (\boldsymbol{x}^\top \boldsymbol{W})^\top \in \mathbb{Y}$ (or $\boldsymbol{w}' = \boldsymbol{W}\boldsymbol{y} \in \mathbb{X}$). The inner product $\langle \boldsymbol{w}', \boldsymbol{y} \rangle$ (or $\langle \boldsymbol{w}', \boldsymbol{x} \rangle$) defines then the preference strength and can be interpreted like the JFPL model parameters in Section 4.2.2.

4.3.5 Discussion

The BilinPL model relates to the JFPL model by means of a joint-feature map specification using the cross-products between dyad member feature vectors. With feature engineering, it could be shown that BilinPL can resemble an existing method for label ranking, namely LinPL, and can extend that model further by the incorporation of additional label descriptions, thus resulting in a model called LinSidePL.

A further extension can be realized with the use of pairwise kernels. With (4.16), an *explicit* definition of the joint-feature map is introduced. In the case where the dimensionality of the objects is higher than the number of training examples, it can be advantageous to refer to an *implicit* representation via *pairwise kernels*. Using the pairwise kernel

$$K_\Phi\left((\boldsymbol{x}', \boldsymbol{y}'), (\boldsymbol{x}'', \boldsymbol{y}'')\right) = \left\langle \Phi(\boldsymbol{x}', \boldsymbol{y}'), \Phi(\boldsymbol{x}'', \boldsymbol{y}'')\right\rangle \quad , \tag{4.41}$$

with the Kronecker product $\Phi(x, y) = \boldsymbol{x} \otimes \boldsymbol{y}$ one is able to rewrite (4.41) using inner products between the $\boldsymbol{x}$ and $\boldsymbol{y}$ vectors or equivalently using domain-specific linear kernels (Basilico and Hofmann, 2004):

$$K_\Phi\left((\boldsymbol{x}', \boldsymbol{y}'), (\boldsymbol{x}'', \boldsymbol{y}'')\right) = \left\langle \boldsymbol{x}', \boldsymbol{x}''\right\rangle \left\langle \boldsymbol{y}', \boldsymbol{y}''\right\rangle = K_x(\boldsymbol{x}', \boldsymbol{x}'') K_y(\boldsymbol{y}', \boldsymbol{y}'') \ .$$

The domain-specific kernels would, in contrast, enable the modeling of non-linear relationships. The learning of such an extended BilinPL model is an interesting aspect for future research.

The BilinPL model with an explicit definition of the joint-feature map, however, relies on *predictive* dyad member feature vectors. Those can be engineered by combining or transforming the attributes of objects. Coming up with good feature vectors is a time-consuming undertaking, and it is tough to engineer them in a way that the bi-linearity assumption would be in accordance with the preference structure of the data. To this end, the PLNet approach is proposed in the next section that offers more flexibility with respect to *learning* joint-feature representations.

4.4 Plackett-Luce Networks

In this section, an alternative dyad ranking approach is described—it is called **PLNet**. It allows for modeling utilities in a more flexible way compared to the BilinPL model from the previous section. Instead of assuming a bilinear structure of the joint-feature representation, the utilities are modeled in PLNet as (feed-forward) neural networks; thanks to the hidden layer of such networks, important non-linear dependencies can thus be captured (Rumelhart et al., 1986). One drawback of the BilinPL model is that it relies on the joint-feature vector formulation that strongly depends on the dyad feature vectors and thus the inductive bias it imposes onto the model can be suboptimal. PLNet instead aims to *learn* the joint-feature vector representation and bypasses the necessity for feature engineering.

4.4.1 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 $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ in a PL model. More specifically, the probability of observing the ranking

$$\rho : (\boldsymbol{x}_1, \boldsymbol{y}_1) \succ (\boldsymbol{x}_2, \boldsymbol{y}_2) \succ \ldots \succ (\boldsymbol{x}_M, \boldsymbol{y}_M)$$

is given by

$$\mathbf{P}\left(\rho \,|\, \boldsymbol{v}\right) = \prod_{m=1}^{M-1} \frac{v_m}{\sum_{l=m}^{M} v_l} = \prod_{m=1}^{M-1} \frac{\exp\left(u_m\right)}{\sum_{l=m}^{M} \exp\left(u_l\right)} \quad , \tag{4.42}$$

that is, by the PL probabilities induced by the parameters

$$v_m = \exp\left(u_m\right) = \exp\left(g(\boldsymbol{x}_m, \boldsymbol{y}_m)\right) \quad .$$

PLNet is a multi-layer perceptron (MLP), and its structure is shown in Figure 4.8. It is a directed graph organized in multiple layers and takes as input two vectors corresponding to the members of a dyad. The layers are denoted by the numbers 1 to L and associated with them are nodes (also called neurons). In this way, the layers exhibit the activation vectors $\boldsymbol{a}^{(l)}$ and the weight matrices $\boldsymbol{W}^{(l)}$. Each node at a particular layer receives the activation values from all the nodes one layer before it. The weight matrix of the layer l hence consists of several column vectors, where each column vector is associated with a

particular node, i, of the layer l. Each component j of the column vector (i.e. the j-th row of the matrix), in turn, refers to the j-th node of the layer $l-1$ before the layer l. These weights are denoted by $w_{j,i}^{(l)} \in \mathbb{R}$ (see Figure 4.7). In the following paragraphs, the superscript in *round* brackets will not be mentioned if the context is clear. A superscript in *rectangular* brackets will be used instead to indicate the affiliation of the weights with a particular network.

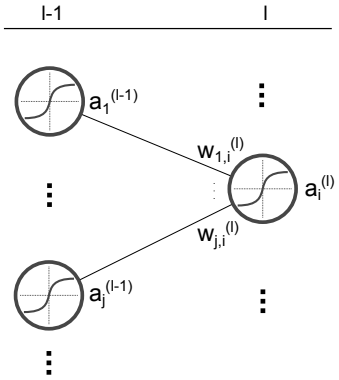

Figure 4.7: PLNet layer weights and activations.

The construction exhibits some special properties. The first layer has nodes that correspond to the features of the input data. Hence, there are no weights associated with them and the activations $\boldsymbol{a}^{(1)}$ are identical to the features. There is at least one hidden layer in the architecture which has nodes using a sigmoidal activation function. The output layer, in contrast, consists of only a single node whose activation produces a scalar value using a linear activation function. Since there is only one node at the last layer, the weight matrix of it is a single vector which is denoted by $\boldsymbol{w}^{(L)} := \boldsymbol{W}^{(L)}$. The values produced by this node are, thus, of the form $u = \langle \boldsymbol{w}^{(L)}, \boldsymbol{a}^{(L-1)} \rangle + b^{(L)}$.

The PLNet architecture shares some resemblance with the Siamese architecture (Bromley et al., 1993; Chopra et al., 2005). In the training phase, the Siamese architecture consists of two identical weight sharing neural networks whose outputs are connected to a (symmetric) function (Bakir et al., 2007). The idea is to use one of both networks after the training to produce a feature representation for a given single input (Nair and Hinton, 2010). Siamese neural networks and extensions are typically applied in computer vision for metric learning, where the training information consists of contrasting pairs of similar and dissimilar images (Bell and Bala, 2015; Han et al., 2015; Veit et al., 2015;

Wang et al., 2018, 2014). A difference between the Siamese architecture and that of the PLNet is that the base network (MLP) of PLNet already takes dyads as input. Another difference is that a *varying* number of identical MLPs are coupled together during the training, as will be explained in the upcoming section.

The basic concepts common to MLPs are shortly reviewed. A node in a network has the task to produce a real-valued output given a set of real valued inputs. These inputs are either data in case of input nodes or else the outputs of other nodes. The inner working of the node j consists of the computation of the *net input* (or *excitation* of the neuron) denoted by $net_j = \langle \boldsymbol{w}^{(j)}, \boldsymbol{a}^{(j-1)} \rangle + b^{(j)}$ (Patterson, 1998; Rojas, 1993). The output a of a node is calculated by applying an activation function, $\sigma : \mathbb{R} \to \mathbb{R}$, on the net input. There are several possibilities for the specification of an activation function. A sigmoidal function is commonly used for the nodes of the hidden layers and refers to a function having a "S"-shaped curve (Bishop, 1995). More concretely, we will primarily use the logistic function

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \ ,$$

when we refer to a sigmoidal function. It is sometimes also known as the "squashing" function because it non-linearly squashes values between zero and one. Another type of activation function is the "linear" function, which is the identity function $\sigma(x) = x$. More recent activation functions, such as the rectified linear unit (ReLU), are possible as well. The usefulness of ReLUs has been shown especially for the training of *deep* neuronal networks (Glorot et al., 2011).

Technically, the bias term $b^{(L)}$ is not needed in the last layer, as it has no effect on the PLNet model (4.42): 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

$$v = \exp\left(\langle \boldsymbol{w}^{(L)}, \boldsymbol{a}^{(L-1)} \rangle + b^{(L)}\right) = \exp(b^{(L)}) \cdot \exp\left(\langle \boldsymbol{w}^{(L)}, \boldsymbol{a}^{(L-1)} \rangle\right) \ , \tag{4.43}$$

and as noted before, the PL model is invariant against the 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 JFPL model (4.7) introduced in Section 4.2. 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

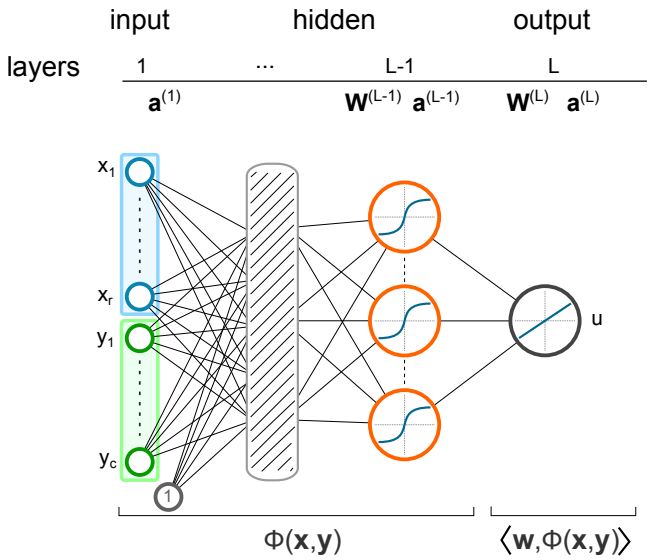

Figure 4.8: PLNet architecture. This kind of feed-forward neural network is composed of several layers. Dyad members $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$ are entered at the input layer, whose nodes are connected to the nodes of the next layer. The inner (hidden) layers have nodes endowed 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.

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 the joint-feature map $\Phi(\boldsymbol{x}, \boldsymbol{y})$.

4.4.2 Training

The training procedure builds upon the basic idea of back-propagation (Rumelhart et al., 1986; Werbos, 1974), 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

$$\rho_n : \left(\boldsymbol{x}_1^{(n)}, \boldsymbol{y}_1^{(n)}\right) \succ \left(\boldsymbol{x}_2^{(n)}, \boldsymbol{y}_2^{(n)}\right) \succ \ldots \succ \left(\boldsymbol{x}_{M_n}^{(n)}, \boldsymbol{y}_{M_n}^{(n)}\right)$$

is described in Listing 6. At the 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

Algorithm 6 PLNet Training

Require: Training examples $\rho_n : \left(\boldsymbol{x}_1^{(n)}, \boldsymbol{y}_1^{(n)}\right) \succ \left(\boldsymbol{x}_2^{(n)}, \boldsymbol{y}_2^{(n)}\right) \succ \ldots \succ \left(\boldsymbol{x}_{M_n}^{(n)}, \boldsymbol{y}_{M_n}^{(n)}\right)$

1: Create M_n copies of master network g_t.
2: Enter $\left(\boldsymbol{x}_k^{(n)}, \boldsymbol{y}_k^{(n)}\right)$ 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: $g_{t+1} \leftarrow \text{update}\left(g_t, \sum_k \Delta w_{j,i}^{[k]}\right)$

$g_{t,k}$. The k-th dyad of ρ_n, when entered into the network $g_{t,k}$, yields the output $u^{[k]}$. Proceeding that way for all $1 \leq k \leq M_n$ dyads, it is possible to evaluate the probability of that ranking according to (4.42). Moreover, we can calculate errors to be used for back-propagation. Recall that the NLL is given by

$$E_n = -\log \mathbf{P}\left(\rho \,|\, \boldsymbol{v}\right) = \sum_{m=1}^{M_n-1} \log \sum_{l=m}^{M_n} \exp\left(u^{[l]}\right) - \sum_{m=1}^{M_n-1} u^{[m]} \ . \tag{4.44}$$

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

$$\delta^{[k]} = \frac{\partial E_n}{\partial u^{[k]}} = \left[\sum_{m=1}^{k} \frac{\exp\left(u^{[k]}\right)}{\sum_{l=m}^{M_n} \exp\left(u^{[l]}\right)}\right] - 1 \ . \tag{4.45}$$

This value can then be used for back-propagation on the k-th 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 k-th network directly, but they are used to carry out the weight adjustments in the *master* network instead. This can either be accomplished by a single update

$$w_{j,i} = w_{j,i} - \sum_k \Delta w_{j,i}^{[k]} \ , \tag{4.46}$$

or by a sequence of updates $w_{j,i} = w_{j,i} - \Delta w_{j,i}^{[k]}$, $1 \leq k \leq M_n$. To justify the accumulation of the weight changes from the individual networks, consider the error term (4.44) as a function of the network outputs, i.e.

$$E_n\left(u^{[1]}(\theta), u^{[2]}(\theta), \ldots, u^{[M_n]}(\theta)\right) \ . \tag{4.47}$$

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 (4.47) with respect to $w_{j,i} \in \theta$, we obtain

$$\frac{\partial E_n}{\partial w_{j,i}} = \frac{\partial E_n}{\partial u^{[1]}} \frac{\partial u^{[1]}}{\partial w_{j,i}} + \frac{\partial E_n}{\partial u^{[2]}} \frac{\partial u^{[2]}}{\partial w_{j,i}} + \ldots + \frac{\partial E_n}{\partial u^{[M_n]}} \frac{\partial u^{[M_n]}}{\partial w_{j,i}} \ . \tag{4.48}$$

The summand $\frac{\partial E_n}{\partial u^{[k]}} \frac{\partial u^{[k]}}{\partial w_{j,i}}$ in (4.48) 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 k-th network's output $u^{[k]}$.

An important point of the procedure is the processing of the training examples in a batch-wise mode at the level of single dyad rankings. The latent utilities $u^{[k]}$, $1 \leq k \leq M_n$ obtained in that way can then be used for the calculation of the log loss (4.44).

The algorithm stops when the error has been diminished without overfitting the data. We suggest *early stopping* as a means for regularization. To this end, the NLL values of the training and validation data are tracked during the whole learning process (Prechelt, 2012). A good point to stop the training and to prevent over-fitting is when the validation NLL value has reached a global minimum. In practical terms, to distinguish between trend from noise, the checks for the event of having found a global minimum does not take place at every iteration but for a certain interval, e.g. every 100 iterations. Finally, predictions can be carried out straightforwardly using a trained PLNet, as described in Section 4.2.1.

4.4.3 Applicability

PLNet can be applied also on the label ranking problem. One possibility to do this is to use the dummy encoding from Section 4.3.2, in which the vectors of the $\mathbb{Y}$ domain are expressed as one-hot vectors. By relaxing the input type from the vector pair $(\boldsymbol{x}, \boldsymbol{y})$ to the single input vector $\boldsymbol{z}$, the PLNet can also be applied on the problem of object ranking (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.

The PLNet master network has been described as a multi-layer perceptron, which takes two feature vectors as inputs. It is, of course, possible and also more common in practice to use it as a building block of a deeper neural architecture. The input layer of

the PLNet can be considered as a concatenation layer of vector representations that are created (or learned) by other parts of a network..

4.4.4 Comparison with Existing NN-based Approaches

An overview of other (preference) learning methods based on neural networks is provided in this section that share some similarities with PLNet.

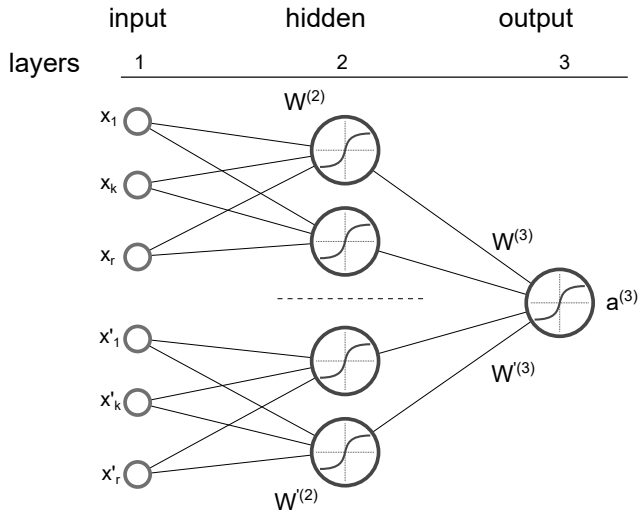

Figure 4.9: Comparison Training architecture.

Comparison training refers to a framework introduced for learning from pairwise preferences with a neural network, originally invented for the game backgammon (Tesauro, 1989). The network architecture consists of two subnetworks, which are connected to a single output node that indicates which of two inputs is preferred over the other (c.f. Figure 4.9). At the training phase, two input vectors of the same dimensionality $\boldsymbol{x}$ and $\boldsymbol{x}'$ are entered at a time together with a target output, which is 1, if $\boldsymbol{x} \succ \boldsymbol{x}'$ and 0 otherwise. The subnetworks are identical in the number of hidden layers, which can be one or more, and they are also equivalent with regard to the weights of their hidden nodes, i.e. $\boldsymbol{W}^{(2)} = \boldsymbol{W}'^{(2)}$. The subnetworks are connected to a single output node in a way that the outputs of the last hidden layer nodes are weighted in the final output node anti-symmetrically, i.e. $\boldsymbol{W}^{(3)} = -\boldsymbol{W}'^{(3)}$. Since both subnetworks are identical, it is sufficient to use just one subnetwork for prediction later on. The architecture provides consistent prediction results because comparisons become unambiguous and transitive.

Both comparison training and PLNet have the idea of using parallel versions of a network and they utilize back-progapation as a learning algorithm. The networks in PLNet, in contrast, are not coupled in a asymmetrical way to support training. But both approaches output real values that can be used to sort vectors in a pairwise fashion in case of comparison training and to sort vectors using their scores directly in the case of PLNet. A major difference is, of course, that PLNet can process rankings as training inputs, whereas comparison training is restricted to pairwise comparisons.

A similar approach called SortNet is proposed by Rigutini et al. (2011). They introduced a three-layered network architecture called CmpNN, which takes two input object vectors and outputs two real-valued numbers. The architecture, provided in Figure 4.10, establishes a preference function for which the properties of reflexivity and symmetry (but not transitivity) are ensured by a weight-sharing technique. More recently, Huybrechts (2016) uses the CmpNN architecture (with three hidden layers) in the context of document retrieval. The CmpNN architecture consists of two subnetworks with

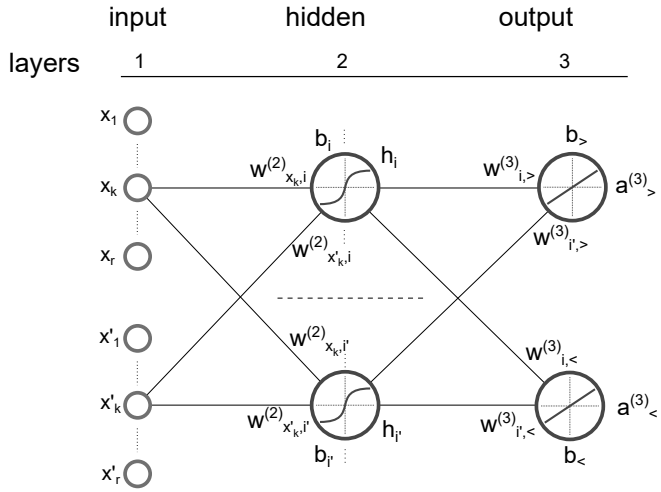

Figure 4.10: CmpNN / SortNet architecture.

three layers that are coupled together so that both input vectors impact the calculations within the subnetworks simultaneously and also the output nodes are influenced by both subnetworks respectively. For each hidden node i in the upper subnetwork, there is a hidden node i' in the lower subnetwork. The coupling is established as follows: First, by demanding $\boldsymbol{w}^{(2)}_{x_k,i'} = \boldsymbol{w}^{(2)}_{x'_k,i}$ and $\boldsymbol{w}^{(2)}_{x'_k,i'} = \boldsymbol{w}^{(2)}_{x_k,i}$, and second, by enforcing $\boldsymbol{w}^{(3)}_{i',>} = \boldsymbol{w}^{(3)}_{i,<}$

and $\boldsymbol{w}^{(3)}_{i',<} = \boldsymbol{w}^{(3)}_{i,>}$. Furthermore, the biases are shared between the hidden nodes and those of the outputs so that $b_i = b'_i$ and $b_> = b_<$. In CmpNN training, the concatenated vector $[\boldsymbol{x}, \boldsymbol{x}']$ is entered together with the output target

$$\boldsymbol{t} = [t_1, t_2]' = \begin{cases} [10]' & \text{if } \boldsymbol{x} \succ \boldsymbol{x}' \\ [01]' & \text{if } \boldsymbol{x}' \succ \boldsymbol{x} \end{cases} .$$

The outputs of the network are denoted by $a^{(3)}_>([\boldsymbol{x}, \boldsymbol{x}'])$ and $a^{(3)}_<([\boldsymbol{x}, \boldsymbol{x}'])$. With them, the network is optimized using training data by minimizing the following squared error function:

$$E([\boldsymbol{x}, \boldsymbol{x}'], \boldsymbol{t}]) = (t_1 - a^{(3)}_>([\boldsymbol{x}, \boldsymbol{x}']))^2 + (t_2 - a^{(3)}_<([\boldsymbol{x}, \boldsymbol{x}']))^2 \ . \tag{4.49}$$

The CmpNN architecture shares some resemblance to PLNet regarding the aspect of using parallel networks. The way they are coupled and the fact that multiple output nodes are present in CmpNN is fundamentally different than in PLNet. Also, the way the inference is established differs. In contrast to PLNet, predictions are possible with CmpNN only on pairs with the rules

$$\begin{cases} \boldsymbol{x} \succ \boldsymbol{x}' & \text{if } a^{(3)}_>([\boldsymbol{x}, \boldsymbol{x}']) \geq a^{(3)}_<([\boldsymbol{x}, \boldsymbol{x}']) \\ \boldsymbol{x} \prec \boldsymbol{x}' & \text{if } a^{(3)}_<([\boldsymbol{x}, \boldsymbol{x}']) \geq a^{(3)}_>([\boldsymbol{x}, \boldsymbol{x}']) \end{cases} .$$

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. That model differs from PLNet in the sense of being tailored for pairwise group comparisons, albeit considering individual skills. The inclusion of feature vectors is of no concern in that model, and the extension to rankings is only suggested for future work.

The label ranking problem, introduced in Section 3.1, had been tackled with neural network approaches previously (Kanda et al., 2012; Ribeiro 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 in the back-propagation procedure. To this end, some variations of this procedure have been investigated. Both previously discussed 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.

In the information retrieval domain, 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 object pairs together with target probabilities, and it is an important difference with PLNet. 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. RankNet considers functions of the form $f : \mathbb{R}^d \to \mathbb{R}$ so that $f(\boldsymbol{z}_i) > f(\boldsymbol{z}_j)$ whenever $\boldsymbol{z}_i$ is judged to be more relevant than $\boldsymbol{z}_j$. A concrete utility function in RankNet can be obtained by training an MLP neural network on pairwise observations using cross-entropy. An important feature of RankNet is that any pair of observation $\{\boldsymbol{z}_i, \boldsymbol{z}_j\}$ has an associated target probability of $\mathbf{P}_{ij} \in [0, 1]$, which is the probability of $\boldsymbol{z}_i$ being ranked higher than $\boldsymbol{z}_j$, or in symbols $\boldsymbol{z}_i \succ \boldsymbol{z}_j$. According to the RankNet authors, these target probabilities could be obtained by measuring the preferences of multiple users on the pairs $\{\boldsymbol{z}_i, \boldsymbol{z}_j\}$.

In order to learn from that kind of data, the RankNet approach optimizes the cross-entropy cost function

$$C_{ij} = -\mathbf{P}_{ij} \log(\hat{\mathbf{P}}_{ij}) - (1 - \mathbf{P}_{ij}) \log(1 - \hat{\mathbf{P}}_{ij}) \ . \tag{4.50}$$

The posterior probabilities $\hat{\mathbf{P}}_{ij}$ are modeled with a (utility) function, f, using the logistic function so that

$$\hat{\mathbf{P}}_{ij} \equiv \frac{\exp(f_{ij})}{1 + \exp(f_{ij})} \ , \tag{4.51}$$

where $f_{ij} := f_i - f_j$ and $f_i := f(\boldsymbol{z}_i)$. With (4.51) it is possible to express (4.50) more compactly as

$$C_{ij} = -\mathbf{P}_{ij} f_{ij} + \log\left[1 + \exp\left(f_{ij}\right)\right] \ . \tag{4.52}$$

The formula (4.52) corresponds to the NLL objective of PLNet under the conditions that all dyad rankings are of the length $K = 2$ and all $\mathbf{P}_{ij}$ are set to one. This can be seen by rearranging (4.52) further:

$$\begin{aligned} C_{ij} &= -\mathbf{P}_{ij} f_i + \mathbf{P}_{ij} f_j + \log\left[1 + \exp\left(f_i - f_j\right)\right] \\ &= -\mathbf{P}_{ij} f_i + \log\left[\exp\left(\mathbf{P}_{ij} f_j\right)\right] + \log\left[1 + \exp\left(f_i - f_j\right)\right] \\ &= -\mathbf{P}_{ij} f_i + \log\left[\exp\left(\mathbf{P}_{ij} f_j\right) + \exp\left(f_i - f_j + \mathbf{P}_{ij} f_j\right)\right] \\ &= -\log\left[\frac{\exp\left(\mathbf{P}_{ij} f_i\right)}{\exp\left(f_i + \mathbf{P}_{ij} f_j - f_j\right) + \exp\left(\mathbf{P}_{ij} f_j\right)}\right] \qquad \square \end{aligned}$$

ListNet (Cao et al., 2007; Luo et al., 2015) similarly uses the cross-entropy as a metric, but in contrast to RankNet, it processes rankings of objects instead of pairwise preferences as training data. There are, however, some important differences between ListNet and the PLNet 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 the PLNet approach, in which non-linear relationships between inputs and outputs can be 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.[5]

4.4.5 Illustration of the Joint-Feature Engineering Problem

In the following, the problem of finding good joint-feature representations is demonstrated. To this end, the PLNet and BilinPL models are compared with each other.

The data used in this illustration has been sampled from a PLNet with three layers. Its inputs are dyads composed of one-dimensional vectors, i.e. $\boldsymbol{x}, \boldsymbol{y} \in [-1, 1]$. The hidden layer has 25 nodes, and all weights are initialized randomly from the uniform distribution

[5] 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.

in $[-15, 15]$. From a total of 400 dyads, 500 training and 50 test rankings were sampled where each comprised five dyads.

For the BilinPL model, we chose from three different variants of input features. They are based on the Kronecker product between object pair features to form joint-feature vectors (resulting in first- and second-order models):

$\boldsymbol{x}_f$	$\boldsymbol{y}_f$	$\Rightarrow \boldsymbol{z}_f$	Identifier
x	y	$[x \cdot y]$	BilinPL-1
$[x, 1]$	$[y, 1]$	$[x, y, xy, 1]$	BilinPL-2
$[x, x^2, 1]$	$[y, y^2, 1]$	$[x, x^2, y, y^2, xy, xy^2, x^2y, x^2y^2, 1]$	BilinPL-3

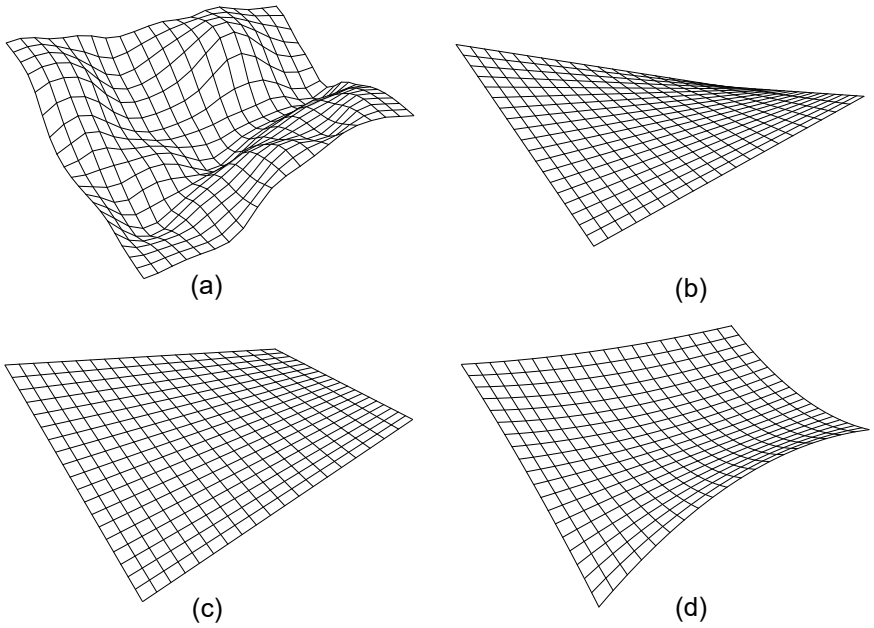

Figure 4.11: Log-skill landscapes of the methods PLNet and BilinPL. (a) is produced by PLNet after learning from dyad rankings. (b)-(d) are produced by BilinPL using different feature specifications.

Table 4.2 and the corresponding Figure 4.11 underpin two key aspects. First, the expressiveness of PLNet can be much larger compared to the used BilinPL versions. Second, the predictive quality varies strongly for BilinPL depending on the choice of dyad features. This is because the usefulness of the joint-feature representation in BilinPL crucially depends on it.

Table 4.2: Results of the synthetic pair ranking data experiment.

	PLNet	BilinPL-1	BilinPL-2	BilinPL-3
Kendall's τ	0.944	0.380	0.644	0.852
Figure 4.11	(a)	(b)	(c)	(d)

4.5 Connections between the Models

JFPL, BilinPL, and PLNet are dyad ranking models, all of which are based on the Plackett–Luce model. They can be considered as generalizations of it because each is capable of modeling dyad rankings, where a dyad is represented as a pair of feature vectors. The modeling of dyad ranking data is accomplished with JFPL on the basis of weighted joint-feature representations. Representations of that kind are realized by means of a joint-feature mapping function that can be applied on dyads.

The joint-feature map aspect can be used to show the interrelationship between the three models. The BilinPL model can be seen as an instance of the JFPL model with the specific choice of a joint-feature map, which is the cross-product between the feature vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ of a dyad. Likewise, the PLNet can also be seen as a JFPL model, whose joint-feature map is represented as part of the network architecture that realizes the mapping.

Table 4.3: Connection between JFPL and other generalized Plackett–Luce models.

Model	Processing	Φ	JFPL/$\log(v)$
BilinPL	calc. cross-products $\Phi(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \otimes \boldsymbol{y}$	bi-linear	$\langle \boldsymbol{w}, \Phi(\cdot, \cdot) \rangle$
PLNet	learn representation $\Phi(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{net}\big(\boldsymbol{\theta}^{+}\big)$	non-linear	$\langle \boldsymbol{w}, \Phi(\cdot, \cdot) \rangle$

The connection between the models can thus be established with different specifications of the joint-feature map $\Phi(\boldsymbol{x}, \boldsymbol{y})$, which is indicated in Table 4.3. The crucial difference between the JFPL model, the BilinPL model, and the PLNet model is that the first two require the engineering of features, while the latter can learn joint-feature representations.

5 Multidimensional Unfolding and Scaling with Dyad Ranking

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 and scaling.

5.1 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 are smaller, the more preferred they are. Unfolding methods produce point configurations in a lower-dimensional space, as illustrated in Figure 5.1. The main assumption in unfolding is that all individuals perceive the world in the same way—this is reflected by one fixed configuration of the items. The differences between individual 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. on 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 the 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

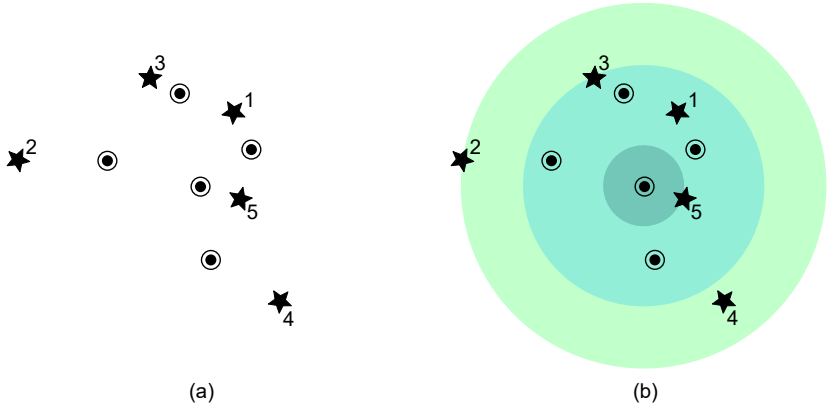

Figure 5.1: Illustration of multidimensional unfolding. Figure (a) shows an unfolding solution, which is a configuration of points: users are represented as dots and items as stars. In Figure (b), the isocontour circles of a particular user are added. They facilitate reading his preference for the items. Here, the preference order is $i_5 \succ i_1 \succ i_3 \succ i_4 \succ i_2$.

lower-dimensional space in such a way that the point distances are in accordance with the distances among the original data points. Kruskal (1964) introduced the squared error function (raw) stress[1], which is the objective that needs to be minimized:

$$\sigma_{\text{raw}}(\boldsymbol{X}) = \sqrt{\sum_{(i,j)} \left(\hat{d}_{i,j} - d_{i,j}(\boldsymbol{X})\right)^2} \ , \tag{5.1}$$

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

$$\hat{d}_{i,j} = f(\delta_{i,j}) \ , \tag{5.2}$$

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

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

following (raw) stress function:

$$\sigma_r^2(\boldsymbol{X}_1, \boldsymbol{X}_2) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left(\hat{d}_{i,j} - d_{i,j}(\boldsymbol{X}_1, \boldsymbol{X}_2) \right)^2 \ . \tag{5.3}$$

5.2 Dyadic Unfolding

With a trained dyad ranking model, it is possible to produce a matrix of latent utilities for pairs of possibly new feature vectors from two domains, for example, from the real-valued vector spaces $\mathbb{X} = \mathbb{X}_1 \times \mathbb{X}_2 \times \cdots \times \mathbb{X}_r$ and $\mathbb{Y} = \mathbb{Y}_1 \times \mathbb{Y}_2 \times \cdots \times \mathbb{Y}_c$. Let $\mathbb{X}^{[\mathrm{tr}]} \subset \mathbb{X}$ be an $n_1 \times r$ matrix and $\mathbb{Y}^{[\mathrm{tr}]} \subset \mathbb{Y}$ an $n_2 \times c$ matrix of feature vectors. Then, the skills $v_{i,j}$ produced by the model on all pairwise vectors of $\mathbb{X}^{[\mathrm{tr}]}$ and $\mathbb{Y}^{[\mathrm{tr}]}$ can be grouped into the 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 $\mathbb{X}$, dissimilarities Δ_Y between objects (feature vectors) from $\mathbb{Y}$, and the preferences on dyads that can be represented by turning their (estimated) PL parameters into dissimilarities Δ_S. A low-dimensional configuration of the points $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ is sought in such a way that the distances of points within $\boldsymbol{X}_1$ are in accordance with the dissimilarities Δ_X, the distances of points within $\boldsymbol{X}_2$ are in accordance with the dissimilarities Δ_Y, and the distances of points between $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are in accordance with Δ_S. The objective of finding the configuration points $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ can be stated as

$$\sigma_D = s(\Delta_X, \Delta_Y, \Delta_S, \boldsymbol{X}_1, \boldsymbol{X}_2) \ . \tag{5.4}$$

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

$$\sigma_D^2 = \alpha \sigma_X^2 + \beta \sigma_Y^2 + \gamma \sigma_S^2 \ , \tag{5.5}$$

with $\alpha + \beta + \gamma = 1$. Expanding (5.5) yields

$$\begin{aligned} \sigma_D^2 = \alpha & \left[\sum_{(i,j)} \left(\hat{d}_{i,j}^{[X]} - d_{i,j}(\boldsymbol{X}_1) \right)^2 \right] + \beta \left[\sum_{(i,j)} \left(\hat{d}_{i,j}^{[Y]} - d_{i,j}(\boldsymbol{X}_2) \right)^2 \right] \\ & + \gamma \left[\sum_{(i,j)} \left(\hat{d}_{i,j}^{[S]} - d_{i,j}(\boldsymbol{X}_1, \boldsymbol{X}_2) \right)^2 \right] \ . \end{aligned} \tag{5.6}$$

Typical formulations of stress, such as (5.1), like the one used in the formulation (5.6), 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 $\mathbb{X}^{[\mathrm{tr}]}$ and $\mathbb{Y}^{[\mathrm{tr}]}$, 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 a major drawback of this formula is that for almost similar items, v requires to become infinitely large. A better alternative, originally also motivated in (Krantz, 1967; Luce, 1961), is $t_2(v_{i,j}) = d_{i,j} = \log(1/v_{i,j}) = -\log(v_{i,j})$. Please note that in the generalized PL models, the $v_{i,j}$ are ensured to be positive, but not necessarily bounded. Therefore, the mapping t_2 is extended by a subsequent affine linear transformation to the unit interval. A further transformation (t_3) is the rank-transformation, which creates rank numbers in descending order of the PL v values.

5.2.1 Utilization of Probabilistic Information

Marden (1995) introduces **Marginals** as a term for a matrix $\widehat{\mathcal{M}}$ that can be used to characterize ranking data. In a sample of N (i.i.d.) rankings, which are all of the length M, it is possible to state an $M \times M$ matrix that is defined as

$$\widehat{\mathcal{M}}(i,k) = \frac{1}{N} \sum_{j=1}^{N} [\pi_j(k) = i] \quad . \tag{5.7}$$

The entry $\widehat{\mathcal{M}}(i,k)$ corresponds to the relative frequency of the item i being ranked at the position k. The i-th row is, therefore, the marginal distribution of the ranks assigned to the item i. And the j-th column is the marginal distribution of the items given the rank j.

Combining unfolding with the PL models offers the possibility to enrich the visualizations with probabilistic information. To this end, consider a particular dyad member, $\boldsymbol{x}_j$ (or ideal point in the unfolding terminology), and a neighborhood of M dyad members $\boldsymbol{y}_i$ with $1 \leq i \leq M$.

The marginals for an unfolding configuration can be obtained for an ideal point, $\boldsymbol{x}_j$, with the PL parameters $\boldsymbol{v}_j = (v_{j,1}, v_{j,2}, \ldots, v_{j,M})^\top$ by calculating the marginal distribu-

tion over the ranks $1 \leq k \leq M$ with

$$\mathcal{M}(i,k) = \sum_{\pi \in \Pi_{k,i}} \mathbf{P}(\pi \,|\, \boldsymbol{v_j}) \ , \tag{5.8}$$

where $\Pi_{k,i} = \{\pi \in \mathbb{S}_M \,|\, \pi(k) = i\}$, i.e. the set of permutations containing the number i at their k-th position. An entry, $\mathcal{M}(i,k)$, obtained with (5.8), corresponds to the probability that $\boldsymbol{y}_i$ is put on the position k in the ranking. Annotating $\boldsymbol{y}_i$ with this distribution (or a part of it, e.g. the most probable positions) provides useful additional information.

As an example, consider a one-dimensional point configuration provided in Figure 5.2. Let the PL parameters for the dyads $\varrho = \{(\boldsymbol{x}, \boldsymbol{y}_1), (\boldsymbol{x}, \boldsymbol{y}_2), (\boldsymbol{x}, \boldsymbol{y}_3)\}$ be $\boldsymbol{v} = (0.1, 0.7, 0.2)^\top$.

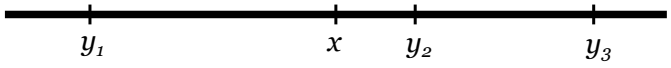

Figure 5.2: One dimensional point configuration for the Marginals example.

The calculation of the probability of $\boldsymbol{y}_2$ being ranked first is then

$$\mathcal{M}(2,1) = \mathbf{P}(231 \,|\, \boldsymbol{v}) + \mathbf{P}(213 \,|\, \boldsymbol{v}) = 0.47 + 0.23 = 0.7 \ .$$

The complete marginals matrix can be calculated similarly and is provided for this example in Table 5.1.

Table 5.1: Marginals. A cell corresponds to the probability of $\boldsymbol{y}_i$ put on rank $1 \leq k \leq 3$.

Rank →	1	2	3
$\boldsymbol{y}_1$	.10	.26	.64
$\boldsymbol{y}_2$	.70	.25	.05
$\boldsymbol{y}_3$	.20	.49	.31

5.3 Dyadic Unfolding with SMACOF

The problem of minimizing (5.5) 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:

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

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

$$\sigma_r^2 = \sigma_r^2(\boldsymbol{X}) = \sum_{i<j} w_{i,j} \left(\hat{d}_{i,j} - d_{i,j}(\boldsymbol{X}) \right)^2 . \tag{5.9}$$

The non-negative weights $w_{i,j}$ in (5.9) 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 into 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, thus enabling 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 (5.9) 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 these, "dyadic" unfolding can be performed by specifying an $(n_1 + n_2)^2$ weight matrix $\boldsymbol{W}$ as

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_{11} & \boldsymbol{W}_{12} \\ \boldsymbol{W}_{21} & \boldsymbol{W}_{22} \end{bmatrix} = \begin{bmatrix} \alpha \boldsymbol{1}\boldsymbol{1}^\top & \gamma \boldsymbol{1}\boldsymbol{1}^\top \\ (\gamma \boldsymbol{1}\boldsymbol{1}^\top)^\top & \beta \boldsymbol{1}\boldsymbol{1}^\top \end{bmatrix} . \tag{5.10}$$

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 PL model parameters that are transformed to express dissimilarity. The main difference with conventional unfolding is, therefore, the inclusion of the within-set proximities.

The iterative procedure employed by SMACOF is described in Listing 7. 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

Algorithm 7 SMACOF

Require: Dissimilarities δ, initial point configuration $\boldsymbol{X}_0$

1: $s_0 \leftarrow \sigma_r(\boldsymbol{X}_0)$, $k \leftarrow 0$
2: **repeat**
3: $\quad k \leftarrow k + 1$
4: $\quad$ Update $\boldsymbol{X}_k$ via the **Guttman transform**
5: $\quad s_k \leftarrow \sigma_r(\boldsymbol{X}_k)$
6: **until** $(s_{k-1} - s_k < \epsilon$ **or** $k = \text{maxiter})$
7: **return** $\boldsymbol{X}_k$

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 (5.9). More details on the derivation can be found in (Borg and Groenen, 2005; De Leeuw and Mair, 2009; Groenen and van de Velden, 2016). The implementation of dyadic unfolding is based on a Matlab port of smacofSym from the "smacof" R package (De Leeuw and Mair, 2009).

5.4 Dyadic Multidimensional Scaling

Dyadic unfolding can be used in conjunction with dyad ranking for the most common case, in which the two domains $\mathbb{X}$ and $\mathbb{Y}$ are different. However, in dyad ranking the case where dyad members stem from a single domain does also exist. In this scenario, an unfolding solution might be hard to interpret because a point can be part of ranking and also be an ideal point at the same time.

Instead, another approach called dyadic multidimensional scaling (DyMDS) is proposed for this case, in which dyads of the form $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are considered. Similar to dyadic unfolding, the DyMDS approach does take the generalized PL models into account and also provides a way to incorporate the dissimilarities of the raw data. We will furthermore require for this approach that the dyads are symmetric so that $(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_j, \boldsymbol{x}_i)$. This is a requisite to apply MDS methods on the v_{ij} scores. The symmetry can be realized, either by enforcing symmetry on the generalized model weights or by post-processing the scores. A simple way of post-processing would be to calculate $V_{sym} = 0.5 \cdot (V^\top + V)$ and to use these as dissimilarity values Δ_S.

The DyMDS algorithm is outlined in Figure 5.3. First, the dissimilarities Δ_S are

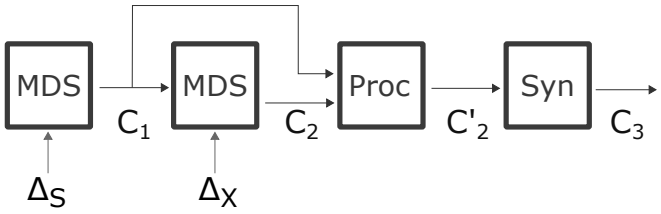

Figure 5.3: Dyadic Multidimensional Scaling (DyMDS) algorithm.

used as inputs for an MDS method. This, in turn, produces the configuration C_1, a set of points of dimensionality p. This configuration is used as a start configuration for the application of a further MDS method. As input for the dissimilarities it takes Δ_X, the pairwise dissimilarities between the $\boldsymbol{x}$ vectors. The outcome of the second MDS application is the configuration C_2. The next step ensures that the orientation of points in C_2 coincides as good as possible with those of C_1. To this end, Procrustes superimposition (Proc) is applied on C_2 and C_1 to generate another configuration, C_2' (Gower, 1975). This is done by translating, rotating, and uniformly scaling the original points in an optimal way, e.g. by the minimization of the sum of squared distances among them.

The last step of the DyMDS is about finding a final configuration, C_3, that allows for the synthesis (Syn) of the point configurations C_1 and C_2'. It requires a single parameter, $\gamma \in [0, 1]$, and similar to dyadic unfolding, it can be understood as a weight that determines the influence of Δ_S on a configuration. As outlined in Diagram 5.4, for each configuration point $\boldsymbol{x}_i \in C_1$ and $\boldsymbol{x}_i' \in C_2'$, a resulting point, $\boldsymbol{x}_i^s \in C_3$, is generated, which is located on a straight line between $\boldsymbol{x}_i$ and $\boldsymbol{x}_i'$ in proportion to γ. More specifically,

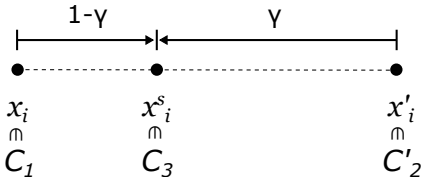

Figure 5.4: Synthesis of two point configurations.

the point $\boldsymbol{x}_i$ is "moved" toward $\boldsymbol{x}'_i$ by $(1-\gamma)$ the amount of the total distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$. In formula, this corresponds to

$$\boldsymbol{x}_i^s = \boldsymbol{x}_i + (1-\gamma) \cdot \left(\boldsymbol{x}'_i - \boldsymbol{x}_i\right) \quad . \tag{5.11}$$

If $\gamma = 1$, the resulting configuration corresponds to C_1, and otherwise if $\gamma = 0$, it corresponds to C'_2.

5.5 Related Visualization Approaches for 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 clusters by utilizing heat map density plots.

An advantage of the newly proposed dyadic unfolding and scaling visualization over this and all other techniques is its capability of providing reliability information on the

distances between the points using a probabilistic ranking model. This is realized by calculating for a chosen ideal point, $\boldsymbol{x}$, and with regard to the (neighboring) $\boldsymbol{y}$ vectors, their respective distribution over the ranks. This information can be used to realize an interactive and enhanced unfolding visualization.

6 Preference-based Reinforcement Learning using Dyad Ranking

Reinforcement learning (RL) is an established machine learning methodology for modeling and optimizing the behavior of an autonomous agent acting in a dynamic environment (Sutton and Barto, 1998). A key component of RL is a numerical reward function, which is used to provide positive or negative (and possibly delayed) feedback signals for the agent's actions. This quest for numerical information impedes the use of RL in situations where precise rewards are difficult to specify.

This was the main motivation for *preference-based reinforcement learning* (PBRL), which has recently been introduced as a generalization of conventional RL (Akrour et al., 2011; Cheng et al., 2011). Instead of numerical rewards, it assumes weaker feedback in the form of qualitative preferences between states or trajectories. A specific realization of preference-based reinforcement learning is a combination of *approximate policy iteration* (Dimitrakakis and Lagoudakis, 2008) and a preference-learning method called *label ranking* (Vembu and Gärtner, 2010). Roughly speaking, label ranking is used to generalize training information of the form "in state $\boldsymbol{s}$, taking action $\boldsymbol{a}$ appears to be better than action $\boldsymbol{a}'$," so that a ranking of all available actions can be predicted for all states of the agent's state space.

In this chapter, we propose an extension of this method in which label ranking is replaced by *dyad ranking*. The main advantage of this extension is the ability of dyad ranking to learn from feature descriptions of actions, i.e. properties of the actions $\boldsymbol{a}$, which are often available in reinforcement learning. This is not possible in standard label ranking, where choice alternatives are merely identified by their name, but not characterized in terms of attributes. Our speculation is that exploiting feature descriptions will improve learning by generalizing, not only over the state space, but also over the action space.

6.1 Reinforcement Learning

Conventional reinforcement learning assumes a scenario in which an agent moves through a (finite) *state space* S by repeatedly selecting *actions* from a set $A = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k\}$. A Markovian *state transition* function $\delta : S \times A \longrightarrow \mathbf{P}(S)$, where $\mathbf{P}(S)$ denotes the set of probability distributions over S, randomly takes the agent to a new state, depending on the current state and the chosen action. Occasionally, the agent receives feedback on its actions in the form of a reward signal, $r : S \times A \longrightarrow \mathbb{R}$, where $r(\boldsymbol{s}, \boldsymbol{a})$ is the immediate reward the agent receives for performing the action $\boldsymbol{a}$ in the state $\boldsymbol{s}$. The goal of the agent is to choose its actions so as to maximize its expected total reward.

The most common task is to learn a *policy* $\pi : S \longrightarrow A$ that prescribes the agent how to act optimally in each situation (state). More specifically, the goal is often defined as maximizing the expected sum of rewards (given the initial state $\boldsymbol{s}$), with future rewards being discounted by the factor $\gamma \in [0, 1]$:

$$V^{\pi}(\boldsymbol{s}) \;=\; E\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t)) \,|\, \boldsymbol{s}_0 = \boldsymbol{s}\right] \tag{6.1}$$

where $(\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \ldots)$ is a trajectory of π through the state space. With $V^*(\boldsymbol{s})$ the best possible value that can be achieved for (6.1), a policy is called optimal if it achieves the best value in each state $\boldsymbol{s}$. Thus, one possibility to learn an optimal policy is to learn an evaluation of states in the form of a value function (Sutton, 1988), or to learn a so-called Q-function that returns the expected reward for a given state-action pair (Watkins and Dayan, 1992):

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \cdot V^{\pi}(\delta(\boldsymbol{s}, \boldsymbol{a}))$$

6.1.1 Approximate Policy Iteration (API)

Instead of determining optimal actions *indirectly* through learning the value function or the Q-function, one may try to learn a policy *directly* in the form of a mapping from states to actions. A particularly interesting approach in this regard is *approximate policy iteration* (API) with rollouts (Dimitrakakis and Lagoudakis, 2008; Lagoudakis and Parr, 2003). The key idea behind this approach is to use a generative model of the underlying process to perform simulations that, in turn, allow for approximating the value of an action $\boldsymbol{a}$ in a given state $\boldsymbol{s}$. To this end, the action is performed, resulting in the state $\boldsymbol{s}_1 = \delta(\boldsymbol{s}, \boldsymbol{a})$. The value of this state is estimated by performing so-called *rollouts*, i.e.

by repeatedly selecting actions following the policy π for at most T steps, and finally, accumulating the observed rewards. This is repeated several times, and the average reward over the rollouts is returned as an approximate Q-value $\tilde{Q}^\pi(\boldsymbol{s}, \boldsymbol{a})$ for taking the action $\boldsymbol{a}$ the state $\boldsymbol{s}$ (leading to $\boldsymbol{s}_1$) and following the policy π thereafter.

The rollouts are then used in a policy iteration loop, which iterates through each of the sample states, simulates all actions in a state, and determines the action $\boldsymbol{a}^*$ that promises the highest Q-value. If $\boldsymbol{a}^*$ is significantly better than all alternative actions in this state, a training example, $(\boldsymbol{s}, \boldsymbol{a}^*)$, is added to the training set $\mathcal{T}$, suggesting that $\boldsymbol{a}^*$ is the best action to take in the state $\boldsymbol{s}$. Eventually, $\mathcal{T}$ is used for a *policy generalization* step, i.e. to induce the state-action mapping $S \longrightarrow A$ that forms the new policy π'; the underlying problem to be solved is a standard (multi-class) classification problem. This process is repeated several times until some stopping criterion is met (e.g. if the policy does not improve from one iteration to the next).

6.2 Preference-based Reinforcement Learning

The key idea of *preference-based reinforcement learning* (PBRL) is to replace the (quantitative) evaluation of individual actions by the (qualitative) comparison between pairs of actions (Cheng et al., 2011; Fürnkranz et al., 2012). Comparisons of that kind are in principle enough to make optimal decisions. Besides, they are often more natural and less difficult to acquire, especially in applications where the environment does not provide numerical rewards in a natural way.

The basic piece of information we consider is a pairwise preference of the form $\boldsymbol{a}_i \succ_{\boldsymbol{s}} \boldsymbol{a}_j$, or, more specifically, $\boldsymbol{a}_i \succ^\pi_{\boldsymbol{s}} \boldsymbol{a}_j$, suggesting that in the state $\boldsymbol{s}$, taking action $\boldsymbol{a}_i$ (and following policy π afterwards) is better than taking action $\boldsymbol{a}_j$.

Evaluating the trajectory $t = (\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \ldots)$ in terms of its (expected) total reward (6.1) reduces the trajectories in comparison with real numbers; thus, comparability is enforced and a total order on trajectories is induced. More generally, and arguably more in line with the idea of qualitative feedback, one may assume a partial order relation $\sqsupset$ on trajectories, which means that the trajectories t and t' can also be incomparable. A contextual preference can then be defined as follows:

$$\boldsymbol{a}_i \succ^\pi_{\boldsymbol{s}} \boldsymbol{a}_j \quad \Leftrightarrow \quad \mathbf{P}(t(\boldsymbol{a}_i) \sqsupset t(\boldsymbol{a}_j)) > \mathbf{P}(t(\boldsymbol{a}_j) \sqsupset t(\boldsymbol{a}_i)) \ , \tag{6.2}$$

where $t(\boldsymbol{a}_i)$ denotes the (random) trajectory produced by taking action $\boldsymbol{a}_i$ in the state $\boldsymbol{s}$ and following π thereafter, and $\mathbf{P}(t \sqsupset t')$ is the probability that the trajectory t is preferred to t'.

6.2.1 API with Label Ranking

In (Fürnkranz et al., 2012), preference-based reinforcement learning is realized in the form of a preference-based variant of API, namely a variant in which, instead of the classifier $S \longrightarrow A$, a so-called *label ranker* is trained for policy generalization. In the problem of label ranking, the goal is to learn a model that maps instances to rankings over a finite set of predefined choice alternatives (Vembu and Gärtner, 2010). In the context of PBRL, the instance space is given by the state space S, and the set of labels corresponds to the set of actions A. Thus, the goal is to learn a mapping

$$S \longrightarrow \Pi(A)\,,$$

which maps states to the total orders (permutations) of the available actions A. In other words, the task is to learn a function that can rank all available actions in a state according to their preference (6.2).

More concretely, a method called *ranking by pairwise comparison* (RPC) is used for training a label ranker (Hüllermeier et al., 2008). RPC accepts training information in the form of binary (action) preferences $(\boldsymbol{s}, \boldsymbol{a}_k \succ \boldsymbol{a}_j)$, indicating that in the state $\boldsymbol{s}$, the action $\boldsymbol{a}_k$ is preferred to the action $\boldsymbol{a}_j$. Information of that kind can be produced thanks to the assumption of a generative model, as described in Section 6.1.1. Subsequently, we refer to this approach as API-LR.

6.3 PBRL using Dyad Ranking

In comparison with the original, the classification-based approach to approximate policy iteration (Section 6.1.1), the ranking-based method outlined in Section 6.2.1 exhibits several advantages. For example, pairwise preferences are much easier to elicit for training than examples of the unique optimal actions $\boldsymbol{a}^*$. Besides, the preference-based approach better exploits the gathered training information—for example, it utilizes the pairwise comparisons between any two actions.

In both approaches, however, the actions $\boldsymbol{a}_i$ are treated as distinct elements with no relation to each other; indeed, neither classification nor label ranking does consider any structure on the set of classes A (apart from the trivial discrete structure). Yet, if classes are actions in the context of RL, A is often equipped with a non-trivial structure because actions can be described in terms of properties/features and can be more or less similar to each other. For example, if an action is an acceleration in a certain direction, as in the mountain car problem (see Section 6.4 below), then "fast to the right" is obviously more similar to "slowly to the right" than to "fast to the left".

Needless to say, the exploitation of feature descriptions of actions is a possible way to improve learning in (preference-based) RL, and to generalize, not only over the state space S, but also over the action space A. It may allow, for example, to predict the usefulness of actions that have never been tried before. To realize this idea, we will make use of *dyad ranking* in the way it is explained next.

6.3.1 API with Dyad Ranking

We are now ready to introduce approximate policy iteration based on dyad ranking (API-DR) as a generalization of API-LR. The former is quite similar to the latter, except that a dyad ranker is trained instead of a label ranker. To this end, training data is again produced by executing a number of rollouts on states, starting with a specified action and following the current policy; see Algorithm 8.

In addition to the representation of actions in terms of features, API-DR has another important advantage. Thanks to the use of the (bilinear) PL model, it is not only able to predict a presumably best action in each state, but also informs about the degree of confidence in that prediction. More specifically, it provides a complete probability distribution over all rankings of actions in each state. Information of this kind is useful for various purposes, as will be discussed next.

6.3.1.1 Exploration versus Exploitation

The rollout procedure (Algorithm 9) is invoked by the subroutine *Evaluate Dyad Ranking* (Line 6 of Algorithm 8). And there the PL model is used in its role as a policy, which means that it has to prescribe a single action, $\boldsymbol{a}^*$, for each state $\boldsymbol{s}$. The most obvious

Algorithm 8 Approximate Policy Iteration based on Dyad Ranking

Require: sample states $\mathcal{S}$, initial (random) policy π_0, max. number of policy iterations p, subroutine **Evaluate Dyad Ranking** for determining dyad rankings for a given state and a set of permissible actions in that state.

function API-DR($\mathcal{S}, \pi_0, p$)
  $\pi \leftarrow \pi_0$, $i \leftarrow 0$
  repeat
    $\pi' \leftarrow \pi$, $\mathcal{D} \leftarrow \emptyset$
    for all $s \in \mathcal{S}$ **do**
      $\rho_s \leftarrow$**Evaluate Dyad Ranking** $(\mathcal{A}(s), \pi)$
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{\rho_s\}$
    end for
    $\pi \leftarrow$ **Train Dyad Ranker** $(\mathcal{D})$, $i \leftarrow i + 1$
  until Stopping Criterion (π, p)
  return π
end function

approach is to compute, for each action $\boldsymbol{a}$, the probability

$$\mathbf{P}(\boldsymbol{a} \,|\, \boldsymbol{W}, \boldsymbol{s}) = \frac{\exp(\boldsymbol{s}^\top \boldsymbol{W} \boldsymbol{a})}{\sum_{i=1}^{K} \exp(\boldsymbol{s}^\top \boldsymbol{W} \boldsymbol{a}_i)} \tag{6.3}$$

of being ranked first and to choose the action maximizing this probability.

Adopting the presumably best action in each state corresponds to pure *exploitation*. It is well known, however, that successful learning requires a proper balance between *exploration* and *exploitation*. Interestingly, our approach suggests a very natural way of realizing such a balance simply by selecting each action $\boldsymbol{a}$ according to its probability (6.3).

As an aside, we note that a generalization of the top-1 variant of the BilinPL model is known in the literature as a softmax called the *Boltzmann exploration* (Cesa-Bianchi et al., 2017; Kuleshov and Doina, 2010). It can be used to control the degree of exploration in a more flexible way:

$$\mathbf{P}(\boldsymbol{a} \,|\, \boldsymbol{W}, \boldsymbol{s}) = \frac{\exp(c \cdot \boldsymbol{s}^\top \boldsymbol{W} \boldsymbol{a})}{\sum_{i=1}^{K} \exp(c \cdot \boldsymbol{s}^\top \boldsymbol{W} \boldsymbol{a}_i)} \tag{6.4}$$

for the constant $c \geq 0$; the larger c, the stronger the strategy focuses on the best actions.

Algorithm 9 Probabilistic Rollout Procedure

Require: Initial state s_0, initial action a_0, policy π, discount factor γ, number of rollouts K, max. length(horizon) of each trajectory L, generative environment model E

function ROLLOUT(π, s_0, a_0, K, L)
  for $k \leftarrow 1$ to K **do**
    while t $< L$ and $\neg$**Terminal State**(s_{t-1}) **do**
      $(s_t, r_t) \leftarrow$ **Simulate**(E, s_{t-1}, a_{t-1})
      $(a_t, p_t) \leftarrow$ **Utilize Policy**(π, s_t)
      $\widetilde{Q}_k \leftarrow \widetilde{Q}_k + \gamma^t r_t$
      $t \leftarrow t + 1$
    end while
    // Remaining rollouts can be skipped if p_t-values are high
  end for
  $\widetilde{Q} \leftarrow \frac{1}{k} \sum_{i=1}^{k} \widetilde{Q}_i$
  return $\widetilde{Q}$
end function

6.3.1.2 Uncertainty Sampling

Another interesting opportunity to exploit probabilistic information is *uncertainty sampling*. Uncertainty sampling is a strategy for active learning in which those training examples are requested and for which the learner appears to be maximally uncertain about (Settles, 2010). In binary classification, for example, these are typically the instances that are located closest to the (current) decision boundary.

In our case, the distribution (6.3) informs about the certainty or uncertainty of the learner regarding the best course of action in the given state $\boldsymbol{s}$ (or alternatively, the uncertainty about the true ranking of all actions in that state). This uncertainty can be quantified, for instance, in terms of the entropy of that distribution or the margin between the probability of the best and the second-best action. Correspondingly, those states can be selected as the sample states $\mathcal{S}$ in Algorithm 8 for which the uncertainty is the highest.

6.4 Standard Benchmarks

To primarily demonstrate the API-DR approach, two well-known problems in RL are tackled: the inverted pendulum problem and mountain car. Instead of being dependent on numerical reward information that is indeed given in these benchmark problems, the API-DR approach can cope with weaker information in the form of preferences. Though not the focus of the upcoming experiments, it is imaginable that an advantage of the approach could be if the numerical information is afflicted with noise.

6.4.1 Inverted Pendulum

The inverted pendulum (also known as *cart pole*) problem (IP) is to balance a pendulum attached on top of a cart. The only way to stabilize the pendulum is by moving the cart, which is placed on a planar ground, to the left or to the right. We adopt the experimental setting from Lagoudakis and Parr (Lagoudakis and Parr, 2003), in which the position of the cart in space is not taken into account (see Figure 6.1 (a)).

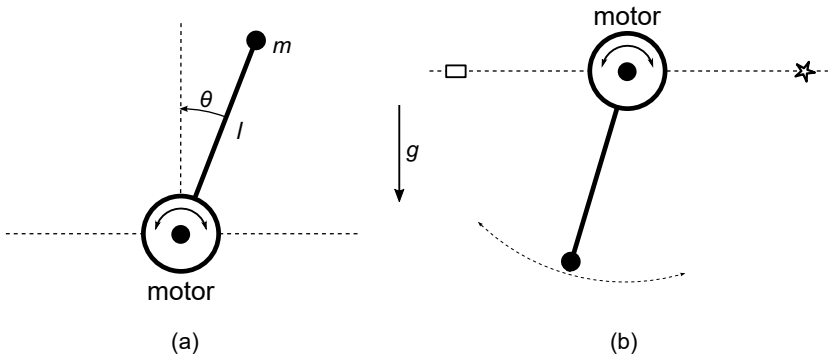

Figure 6.1: Schematic diagrams of (a) the inverted pendulum and (b) the mountain car problem. In (a) the goal is to keep the pendulum close to the vertical axis for a period of time, whereas in (b) an underpowered motor must be used in interaction with the gravity to reach the star.

In the original formulation, three actions are possible, and they are mapped onto the forces of $\{-10, 0, 10\}$ Newtons. The state space is continuous and two-dimensional. The first dimension captures the angle θ between the pole and the vertical axis, whereas the second dimension describes the angle velocity $\dot{\theta}$. Here, the dot notation refers to the first derivative of a function with respect to time. The transitions of the physical model are

Parameter	Symbol	Value	Unit
Gravity	g	9.81	$\mathrm{m/s}^2$
Cart mass	M	8.0	kg
Pendulum mass	m	2.0	kg
Pendulum length	l	0.5	m

Table 6.1: Generative model parameters of inverted pendulum.

determined by the nonlinear dynamics of the system; they depend on the current state $s = (\theta, \dot{\theta})$ and the current action value a, respectively:

$$\ddot{\theta} = \frac{g\sin(\theta) - \alpha ml(\dot{\theta})^2\sin(2\theta)/2 - \alpha\cos(\theta)a}{4l/3 - \alpha ml\cos^2(\theta)},$$

where $\ddot{\theta}$ is the second derivative in time, $\alpha = 1/(m+M)$ and the residual parameters are chosen as in Lagoudakis and Parr (see Table 6.1).

6.4.2 Mountain Car

The mountain car problem (MC) consists of driving an underpowered car out of a valley (see Figure 6.1 (b) for a schematic diagram). The agent must learn a policy that considers the momentum of the car when driving the car along the valley sides. It can basically power or throttle forward and backward. At each time step, the system dynamics depend on the state $s_t = (x_t, \dot{x}_t)$ and the action a_t. It is described by the following equations:

$$\begin{aligned} x_{t+1} &= b_1(x_t + \dot{x}_{t+1}) \\ \dot{x}_{t+1} &= b_2(\dot{x}_t + 0.001a_t - 0.025\cos(3x_t)), \end{aligned}$$

where b_1 is a function that restricts the position x to the interval $[-1.2, 0.5]$ and b_2 restricts the velocity to the interval $[-0.07, 0.07]$. In case the agent reaches $x_t = -1.2$, an inelastic collision is simulated by setting the velocity $\dot{x}$ to zero. The gravity depends on the local slope of the mountain, which is simulated with the term $0.025\cos(3x_t)$. As long as the position x is less then 0.5, the agent receives zero reward. If the car hits the right bound ($x = 0.5$), the goal is achieved, the episode ends, and the agent obtains Reward 1.

In both problems, the actions are simulated to be noisy—this results in non-deterministic state transitions. Thus, the learner is required to perform multiple rollouts. In particular, we add random noise from the intervals $[-0.2, 0.2]$ and $[-0.01, 0.01]$ to the raw action signals for IP and MC, respectively.

6.4.3 Experimental Results

Full and partial action set evaluations We hypothesize that the incorporation of action features can improve the quality of learned policies, especially in situations where data is scarce. To this end, the quality the policies learned by API-LR[1] and API-DR are measured under different conditions. We chose a moderate number of 17 actions on both environments by dividing the original number range into 17 equally sized parts. Thus, the action spaces for IP and MC are given, respectively, as follows:

$$A_{IP} = \{-10, -8.75, -7.5, -6.25, \ldots, 10\}$$
$$A_{MC} = \{-1, -.875, -.75, -.625, \ldots, .875, 1\} \ .$$

Recall that while API-DR can interpret the actions as numbers and hence hence to exploit the metric structure of the real numbers, API-LR merely considers all actions as distinct alternatives.

We furthermore defined three conditions referred to as *complete*, *partial*, and *duel*. Under the first condition, preferences about the entire action set are available per state. In the *partial* condition, the learner can only learn from three randomly drawn actions per state. In the last condition, only two actions are drawn, leading to only one preference per state. Under all conditions, the number of the sampled states $|\mathcal{S}|$ was set fixed to 50 for the MC task and 100 for the IP task. For evaluation, we follow (Cheng et al., 2011; Dimitrakakis and Lagoudakis, 2008) by plotting the cumulative distribution of success rates over a measure of complexity, i.e. the number of actions needed throughout the API procedure for generating a policy that completes a task successfully. In MC, a task is solved successfully if the car has reached the goal at the top of the mountain as fast as possible, i.e. with the fewest number of steering decisions. Opposed to that, the task is tackled successfully in IP if the pendulum has not fallen down within a period of time, i.e. a horizon of 700 (time) steps in the experiments.

[1] Throughout all experiments we used the RPC method with logistic regression as the base learner.

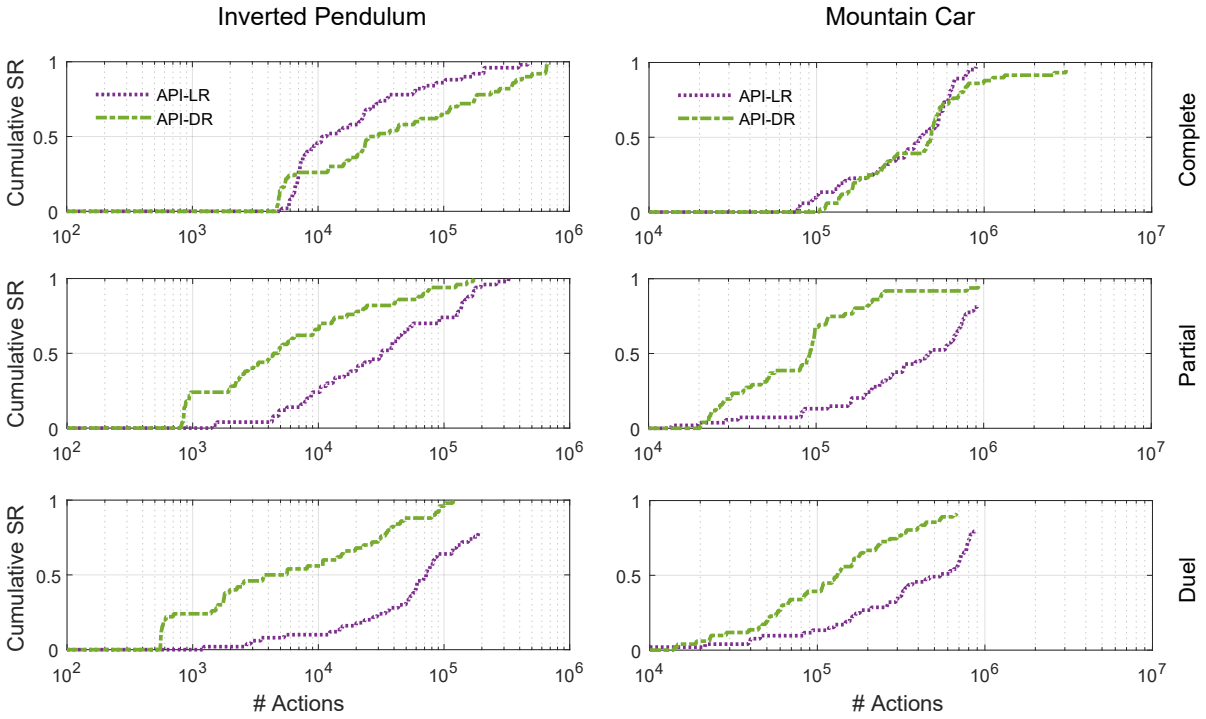

Figure 6.2: Performance of the methods for the inverted pendulum (first column) and the mountain car task (second column).

In the case of the MC/IP tasks, the number is obtained by summing up the average numbers of actions performed for each of the K rollouts realized on initial (state, action) pairs[2]. The point (x, y) in these plots can be interpreted as the minimum number of actions, x, required to reach a success rate of y.

The results are provided in Figure 6.2. It can be seen that the performance of API-LR and API-DR are almost on par with a slight advantage of API-LR at the complete scenario. For all other scenarios with partial information, the API-DR is superior to API-LR, where the use of action values has a positive effect on the performance.

Action set transfer Another important advantage of modeling side-information on actions is the ability to make predictions on actions that were not present in the training phase (c.f. ZSL (Larochelle et al., 2008; Palatucci et al., 2009)). In ZSL, predictions

[2]Note that the number of actions is not fixed per rollout and rather depends on the quality of the current policy. This includes the case that rollouts can stop prematurely before the maximal trajectory length, L, is reached.

must be carried out on class labels that were not present during training. With API-DR, it is possible to do similarly a policy prediction on actions that were unknown before. In terms of *transfer-learning*, the BilinPL model is learned using the action set A_0 (source task) and the trained model is used later on in environments where the action sets are composed differently (target tasks) (Lazaric, 2012). The modified action sets that we consider simulate a certain condition of the control unit motor; these are:

A_1 - *Motor Failure*: intensity levels of actions diminished or even missing,

A_2 - *Motor Boost*: contains additional actions with higher intensity levels.

A trained policy for the MC task applied on the action sets A_0, A_1 and A_2 can be seen in Figure 6.3. The learned policy manages to achieve the goals successfully under all different action set compositions.

Efficiency of Uncertainty Rollouts In this experiment, the benefit of using "uncertainty rollouts," as described in Section 6.3.1.2, is investigated. To this end, API-DR is used with the BilinPL model in two different ways on the inverted pendulum problem. This first version takes uncertainty into account, whereas the second does not. The first has the capability to skip those rollouts about which the uncertainty is low. As evaluation metric, the *absolute* numbers of actions performed during the rollouts are used. The result is shown in Figure 6.4 and suggests that the uncertainty rollout technique can help to improve the runtime performance of API-DR.

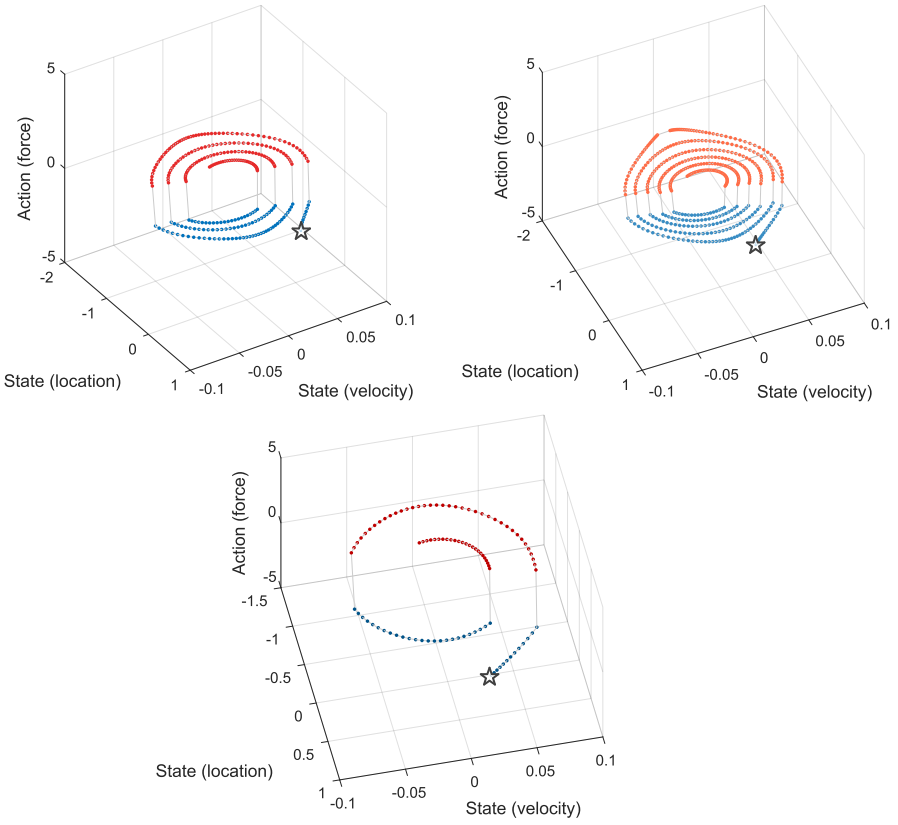

Figure 6.3: Trajectories of a policy visualized in state space of the mountain car problem beginning at the same state $s_0 = (-0.65, -0.01)$. Figure (a) shows a policy on the action set A_0, whereas Figures (b) and (c) show trajectories using the action sets A_1 and A_2, which correspond to the motor "failure" and "boost" scenarios.

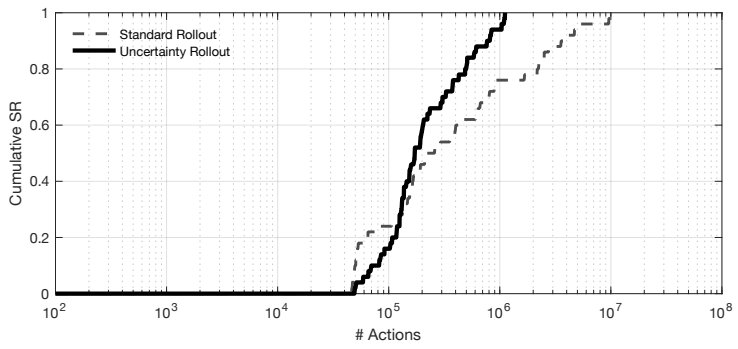

Figure 6.4: API-DR with and without *uncertainty rollouts*.

7 Experiments on Dyad Ranking

To evaluate the performance of the dyad ranking methods BilinPL and PLNet, as well as the usefulness of the dyad ranking setting itself, a number of experiments are conducted.

- In the first part, the advantage of dyad ranking over label ranking is investigated using learning curves on synthetic data. This is followed by a series of experiments with semi-synthetic data. The learned models are used, in turn, to demonstrate the usefulness of the dyadic unfolding method.

- The second part is about the application of dyad ranking for meta-learning. The advantages of taking the label feature description into account is also a major topic in this part. Furthermore, the zero-shot learning aspect is also investigated in this context.

- As a proof of concept, a dyadic unfolding solution is created on a modeled multi-label classification problem. To this end, PLNet is trained using the calibrated label ranking technique.

- In the fourth part, approximate policy iteration with dyad ranking is used as a preference-based reinforcement learning approach for the application of determining image-processing pipeline configurations.

- The fifth, and the last part, is about the application of dyad ranking for similarity learning on images. In this part, the special case of dyad ranking is interesting—here the feature vectors stem from a common domain. Moreover, the modeled similarity is visualized using DyMDS.

In addition to BilinPL and PLNet, the LinPL model (as implemented by Cheng et al. (2010)) and the following state-of-the-art label ranking methods are used in the first two parts 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 the base learner[1]. For comparing with conditional ranking, the QueryRankRLS method is used as implemented in the software package RLScore (Pahikkala and Airola, 2016). BilinPL and PLNet are realized 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., 2016).

7.1 Comparison with Label Ranking

7.1.1 Learning Curves on Synthetic Data

The goal of the first experiment is to identify the conditions under which the use of label attributes are advantageous. Ideal synthetic ranking data is created to this end by sampling from the Plackett–Luce distribution according to the BilinPL model specification under the setting (2.6) of contextual dyad ranking. A realistic scenario is simulated in which $\boldsymbol{y}$ vectors can be missing, i.e. the observed rankings are incomplete. In terms of label ranking this means that labels can be missing. The connection between vectors and labels is provided by a bijection between a collection of vectors and a collection of labels. In terms of dyad ranking, there is a collection of the vectors $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_{,3}, \ldots\}$, whereas label ranking methods just use the labels $\{\lambda_1, \lambda_2, \lambda_3, \ldots\}$. Thus, the vector $\boldsymbol{y}_i$ can be considered as a label description for the label λ_i.

To simulate missing labels, a biased coin is flipped for every vector $\boldsymbol{y}_i$, and it is decided with the probability $p \in [0, 1]$ to keep it or to delete it. A missing rate of $p = 0.3$ was chosen, which means that on average 70% of all vectors of the training set are kept, while the remaining vectors are removed. The feature vectors of the dimensionality $c = 4$ for the $\boldsymbol{y}$ vectors and the dimensionality $r = 3$ for the instance vectors $\boldsymbol{x}$ were generated by sampling the elements from a standard normal distribution (except for one constant instance feature to account for a bias). 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 it is furthermore averaged over 10 repetitions. Recall that CC, LinPL, and RPC are pure label ranking methods that cannot exploit any attribute information on labels.

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

More specifically, the data is created using three nested loops. In Loop 1, features are sampled for each vector $\boldsymbol{y}_i$, $1 \leq i \leq M$, and are kept fixed for the other inner loops. In Loop 2, the data is generated for the different folds by creating the ground truth bilinear matrix underlying the ranking distributions and the test set; it is kept fixed for the next inner loop. In the third and last loop, training sets of different sizes are produced—this includes the generation of the new instance vectors $\boldsymbol{x}$ and their corresponding orderings over the $\boldsymbol{y}$ vectors.

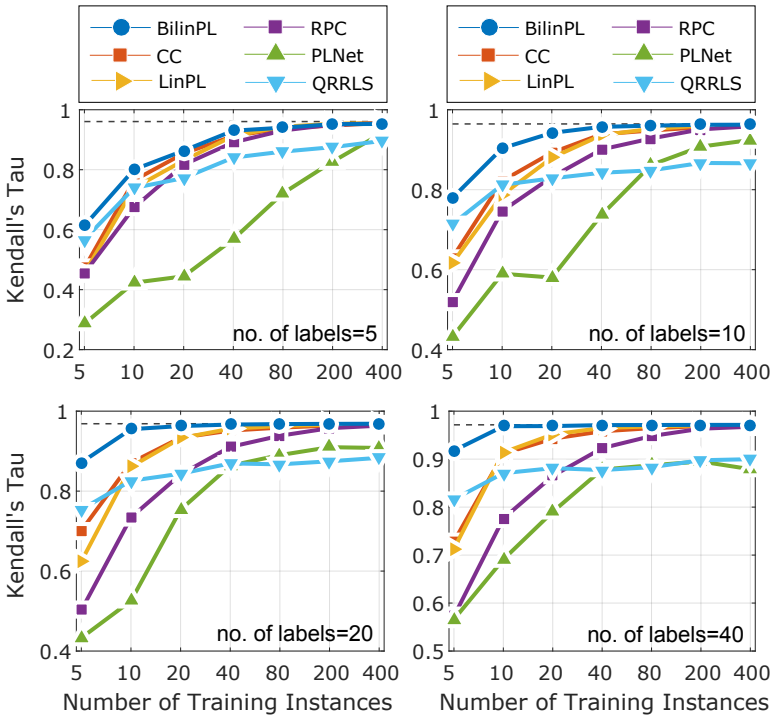

Figure 7.1: Learning curves (generalization performance as a function of the number of training examples) of the ranking methods for different numbers of labels.

The learning curves are shown in Figure 7.1 for different numbers of $\boldsymbol{y}$ vectors (labels). Overall, all ranking methods can learn and predict correctly if enough training data is available. In the limit, they all reach the performance of the "ground truth": Given complete knowledge of 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.

7.1.2 Benchmark on Semi-Synthetic UCI Data

A compilation of datasets were compiled recently to assess the performances of label ranking algorithms (Cheng et al., 2009). The data sets are semi-synthetic in the sense of being based on real multi-class and regression datasets taken from the UCI Machine Learning Repository (Asuncion and Newman, 2007; Lichman, 2013), and these are modified so as to fit the setting of label ranking; see Table 7.1 for the main properties of such data. For the multi-class datasets, rankings were generated by training a naive Bayes classifier on the complete dataset and ordering the class labels according to the predicted class probabilities for each example. For the regression datasets, a subset of 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 7.1: 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	20640	4	4
iris	150	4	3	cpu-small	8192	6	5
pendigits	10992	16	10	elevators	16599	9	9
segment	2310	18	7	fried	40769	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

7.1.2.1 Experimental Results

We compare the performance of BilinPL and PLNet to other state-of-the-art label ranking methods using five-time repeated 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 7.2 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 tendency of overfitting in cases where this flexibility is not needed. This can be seen on datasets with many attributes, but a small number of instances. Linear models are advantageous than PLNet if their inductive bias is correct, which is the case, for example, for the fried data.

7.1.3 Unfolding of Label Ranking Predictions

The PLNet model is used on the example of the vowel dataset to construct an unfolding. Being capable of modeling non-linear relationships, PLNet could outperform all other methods specifically on this dataset. The vowel data consists of 528 instances and 11 labels. The data is split into 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 (5.10).

In Figure 7.2, 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 $L10 \succ L8 \succ L9 \succ L1 \succ L2 \succ \ldots$, which is reflected by the unfolding reasonably well. In the figure, the unfolding is supplemented by a list of top three positions (with corresponding probabilities) for each label according to (5.8). The complete marginal distributions are given in Table 7.3, with the values shown in the visualization highlighted in the bold font.

Table 7.2: Results on the UCI label ranking datasets (average Kendall τ $\pm$ standard deviation). The winning method per dataset is indicated in bold face.

data set	BilinPL	CC	PLNet	QRRLS	RPC-LR
authorship	**0.931**$\pm$**0.013**	0.916$\pm$0.015	0.908$\pm$0.025	0.432$\pm$0.043	0.917$\pm$0.020
bodyfat	0.268$\pm$0.059	0.245$\pm$0.052	0.251$\pm$0.040	0.284$\pm$0.057	**0.285**$\pm$**0.061**
calhousing	0.220$\pm$0.011	0.254$\pm$0.009	**0.272**$\pm$**0.014**	0.215$\pm$0.011	0.243$\pm$0.010
cpu-small	0.445$\pm$0.016	0.468$\pm$0.017	**0.500**$\pm$**0.019**	0.376$\pm$0.012	0.449$\pm$0.016
elevators	0.730$\pm$0.007	0.770$\pm$0.009	**0.788**$\pm$**0.009**	0.570$\pm$0.007	0.749$\pm$0.008
fried	0.999$\pm$0.000	0.999$\pm$0.000	0.951$\pm$0.010	0.996$\pm$0.001	**1.000**$\pm$**0.000**
glass	0.835$\pm$0.072	0.830$\pm$0.079	0.846$\pm$0.080	0.818$\pm$0.075	**0.889**$\pm$**0.057**
housing	0.655$\pm$0.040	0.639$\pm$0.044	**0.703**$\pm$**0.033**	0.579$\pm$0.038	0.672$\pm$0.041
iris	0.813$\pm$0.112	0.800$\pm$0.109	**0.960**$\pm$**0.049**	0.800$\pm$0.064	0.911$\pm$0.047
pendigits	0.892$\pm$0.003	0.896$\pm$0.002	0.905$\pm$0.005	0.561$\pm$0.003	**0.932**$\pm$**0.002**
segment	0.903$\pm$0.008	0.910$\pm$0.008	**0.939**$\pm$**0.008**	0.720$\pm$0.011	0.929$\pm$0.009
stock	0.704$\pm$0.016	0.714$\pm$0.016	**0.882**$\pm$**0.020**	0.663$\pm$0.016	0.774$\pm$0.024
vehicle	0.855$\pm$0.020	0.850$\pm$0.025	**0.872**$\pm$**0.025**	0.776$\pm$0.031	0.855$\pm$0.015
vowel	0.581$\pm$0.026	0.577$\pm$0.046	**0.805**$\pm$**0.016**	0.574$\pm$0.026	0.644$\pm$0.021
wine	0.929$\pm$0.052	0.914$\pm$0.069	**0.942**$\pm$**0.034**	0.923$\pm$0.065	0.925$\pm$0.054
wisconsin	0.629$\pm$0.028	0.612$\pm$0.030	0.514$\pm$0.028	0.630$\pm$0.031	**0.632**$\pm$**0.027**
average rank	3.063	3.438	2.000	4.500	2.000

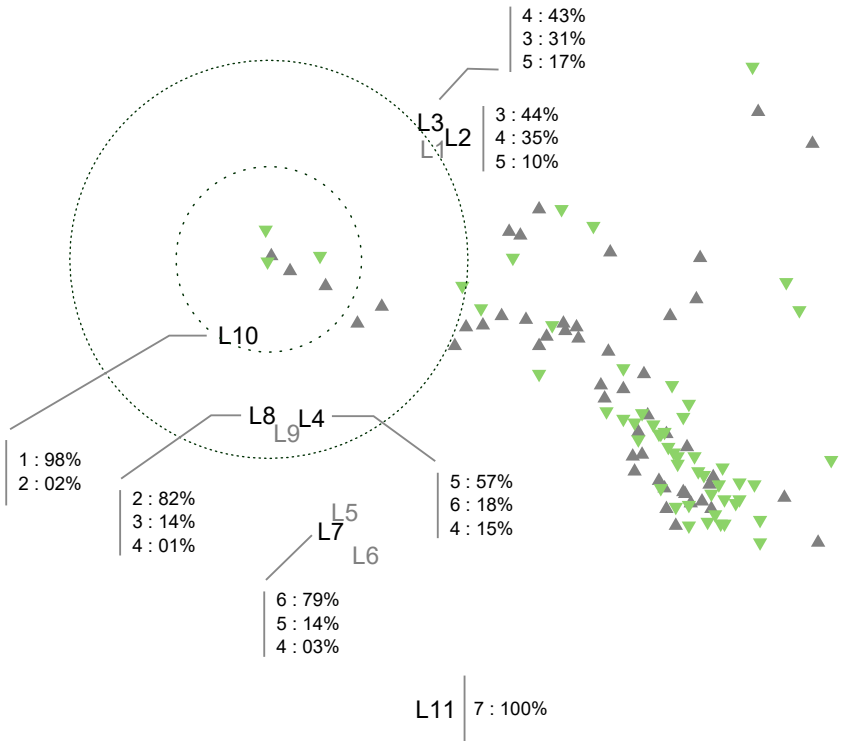

Figure 7.2: Unfolding representation of modeled label rankings of the vowel dataset with PLNet. Triangles denote instances (upright/gray: training, upside-down/green: testing) and class labels are indicated with L[1-11]. The circles indicate isopreference contours for a test instance at their center. 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 far 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 7.3.

Table 7.3: Marginal distributions on a subset of labels for the highlighted test instance in Figure 7.2. The top three probabilities are written in bold-face. For each label, the square brackets indicate the sorting of probabilities in descending order.

		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]**

7.2 Configuration Learning for Genetic Algorithms

In the following experiment, dyad ranking methods are applied within the setting of meta-learning for algorithm recommendation, a setting described in Brazdil et al. (2008). This means, in particular, that for a given problem instance, the aim is to predict a ranking of genetic algorithm (GA) configurations. In terms of dyad ranking, the $\boldsymbol{x}$ vectors correspond to instances (TSP problems) and the $\boldsymbol{y}$ vectors correspond to algorithm descriptions.

The meta-learning framework provides several degrees of freedom, including the way in which meta-data is acquired (see Figure 7.3). The meta-features as part of the meta-data should be able to relate a problem instance (e.g. a dataset) 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 is probing a few parameter settings of the algorithm under consideration (Pfahringer et al., 2000). 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 the algorithm performance.

By analogy with the majority classifier typically used as a baseline in multi-class classification, the meta-learning literature suggests a simple approach called the average

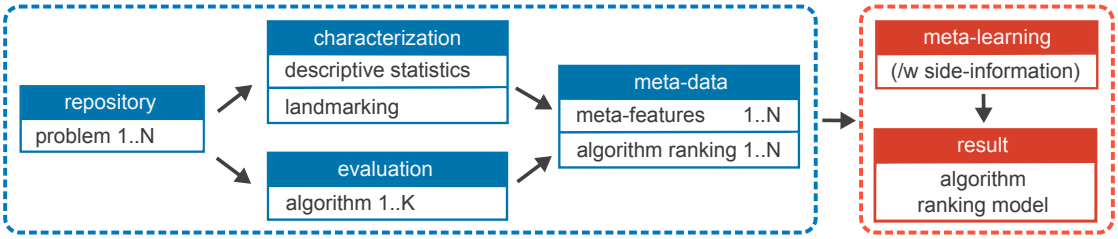

Figure 7.3: The components of the "meta-learning for algorithm recommendation" framework shown above are based on (Brazdil et al., 2008). The left box shows the meta-data acquisition process, which consists of the problem (or dataset) description and the evaluation of the algorithms on the problems (or datasets). 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.

ranks (AR) method (Brazdil et al., 2008). This approach corresponds to 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 the 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). If the data contain rankings in which not all configurations obtained a rank—i.e. the case of incomplete rankings—the AR formula is adapted as follows (Brazdil et al., 2008):

$$\overline{R}_j = \frac{\sum_{i=1}^{N} b_{i,j}}{\sum_{i=1}^{N} \mathbf{1}_{\{R_{i,j} \neq \varnothing\}}}$$

with

$$b_{i,j} = \begin{cases} R_{i,j} & \text{if } R_{i,j} \neq \varnothing \\ 0 & \text{else} \end{cases} ,$$

where $\varnothing$ indicates a missing rank value.

In this experiment, the focus is set 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. All the GAs share the properties of using the same selection criterion, which is "roulette-wheel," the same mutation operator, which is "exchange mutation," and the "elitism" of 10 chromosomes (Mitchell, 1998).

The performance of three groups of GAs are tested on a set of TSP instances. The groups are determined by their choice of the crossover operator, which is 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.

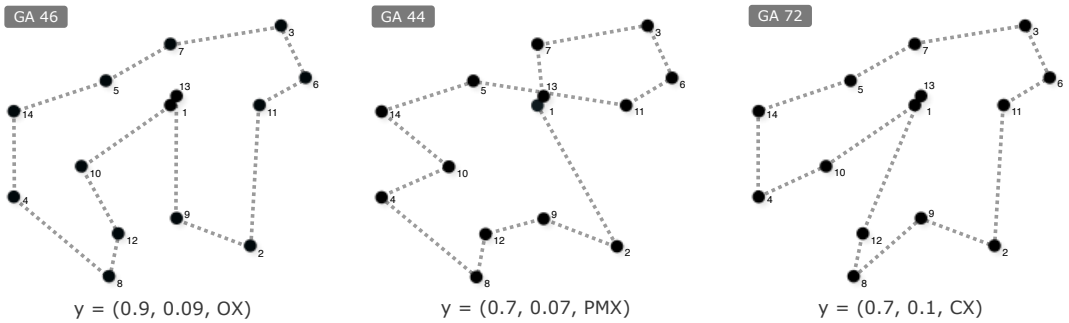

Figure 7.4: Genetic Algorithms are ranked according to the results they produce on the TSP problem. The ground truth ranking in terms of path length in this example is GA44 $\succ$ GA46 $\succ$ GA72.

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 same number of generations, 500, and a population size of 100. The combinations of all the other parameters, namely the crossover type, crossover rate, and the 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. An example of this scenario is provided in Figure 7.4. The label and dyad rankers are faced with rankings under the different conditions (M, N), with N being 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).

7.2.1 Experimental Results

The results in Table 7.4 are quite consistent with the 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 can take advantage of this information for small values of M and compares favorably to the other label rankers (and, in addition, has 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.

7.2.2 Unfolding of Genetic Algorithm Configurations

The BilinPL model is used to create an unfolding solution for the meta-learning problem. To this end, the BilinPL model is trained first on 120 examples, each of which provides an incomplete ranking over only five 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 Figure 7.5, nicely reveals the degree of suitability of GA configurations for particular TSP problems: GAs of the types OX and PMX are more suitable for smaller problems, while GAs of the type CX are better suited for TSPs 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

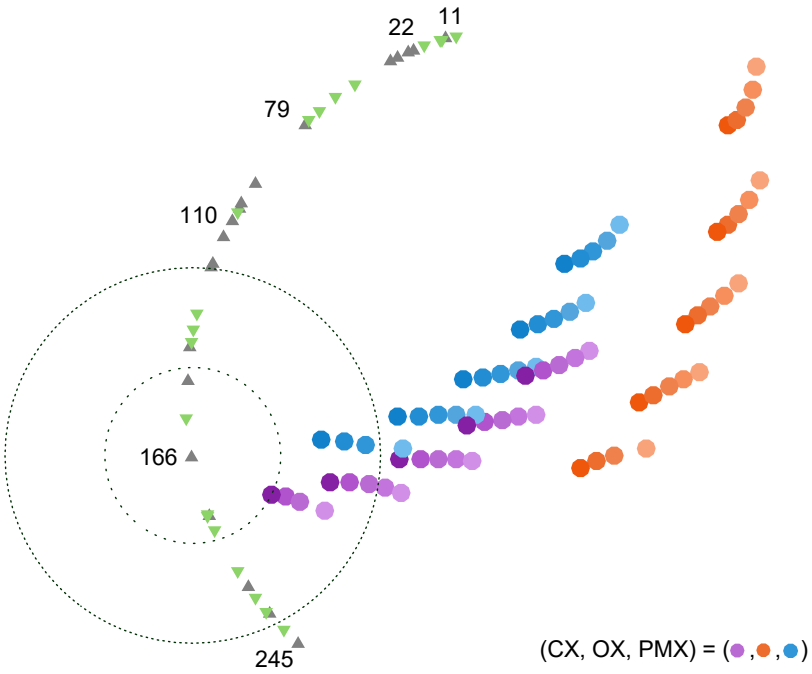

Figure 7.5: Unfolding of GA configuration preferences modeled with BilinPL. Triangles reflect training (upright/gray) and testing (upside-down/green) TSP instances, and circles refer to GA configurations. A random selection of instances are annotated with their associated numbers of cities. Instances are distributed across a curved formation with fewer cities at the 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.

Table 7.4: 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

around a particular instance (here a training problem with 166 cities) to support the inspection of the GA ranking.

7.2.3 Addressing Cold-Start Problems

Additional descriptions in the form of feature vectors are known in the recommender systems domain, where they are typically called *side-information* and are used for tackling *cold-start problems*. These problems refer to situations where preference indicators (e.g. ratings) for new users or new items are not yet available. In these situations, side-information is helpful because it enables to put existing and new entities into relation.

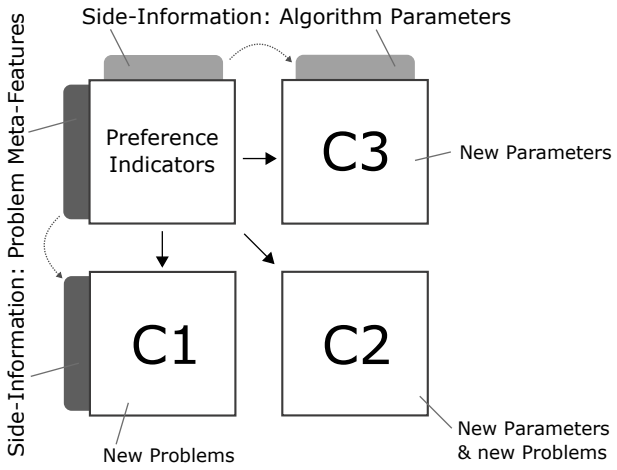

Figure 7.6: Three kinds of cold-start problems are shown. They are characterized in such a way that no preference indicators are available for algorithms or problems. Side-information can help in these situations for inferring preferences and thus recommendations.

In this section, we make use of dyad ranking to predict the rankings of candidate algorithms contextualized by problem instances, assuming that the algorithms exhibit a representation in terms of a feature description. By generalizing over both, attributes of problems as well as algorithms, it becomes possible to tackle cold-start scenarios in which predictions are sought for algorithms that never occurred in the training data. A similar viewpoint is taken in the meta-learning domain by (Misir and Sebag, 2013; Stern et al., 2010), where algorithm recommendation is tackled by means of collaborative filtering (CF) techniques. However, in contrast to the description of users and items in standard CF, side-information for describing problems in meta-learning is usually carefully crafted (Brazdil et al., 2008). As testbed, we present the experimental results on the task of genetic algorithm (GA) recommendation in the cold-start situation corresponding to the box C2 (at the lower right corner) in Figure 7.6.

The (preference) meta-learning dataset for this experiment consists of rankings over 72 different parameterized genetic algorithms (GAs) applied on instances of the traveling salesman problem. The experimental protocol follows a leave-one-out cross validation (LOOCV) procedure over a total number of 246 examples (problems) and 72 GAs description vectors is applied: For the vector $\boldsymbol{y}_j$ ($1 \leq j \leq 72$), the BilinPL model is trained

Table 7.5: Results of the meta-learning cold-start experiment. Rank deviations (data are mean±sd) of the reference (C1) and the cold-start situation (C2) relating to the ground truth are shown. Baseline is Borda Count/Average Ranks in C1.

GA	1	2	3	4	5	6
Baseline	8.41 ± 7.27	9.59 ± 17.34	8.18 ± 7.88	7.19 ± 12.27	5.99 ± 4.33	14.80 ± 10.46
Reference	5.65 ± 4.92	4.09 ± 5.00	6.01 ± 5.21	4.19 ± 5.62	5.11 ± 3.79	6.33 ± 5.96
Cold-start	5.65 ± 4.93	3.98 ± 4.77	6.02 ± 5.16	4.19 ± 5.58	5.16 ± 3.80	6.35 ± 5.95
GA	7	8	9	10	11	12
Baseline	5.28 ± 3.66	16.02 ± 18.59	7.88 ± 5.85	6.72 ± 4.78	11.34 ± 12.27	17.21 ± 11.52
Reference	7.76 ± 5.39	4.69 ± 5.72	6.30 ± 5.74	5.73 ± 5.11	5.66 ± 5.49	6.48 ± 5.15
Cold-start	7.86 ± 5.32	4.70 ± 5.72	6.30 ± 5.73	5.79 ± 5.12	5.67 ± 5.50	6.81 ± 5.21
...						
GA	67	68	69	70	71	72
Baseline	11.79 ± 15.55	11.96 ± 7.56	9.35 ± 7.87	13.56 ± 16.99	13.24 ± 11.40	8.33 ± 7.85
Reference	6.54 ± 5.94	6.44 ± 5.48	8.06 ± 6.10	5.01 ± 5.55	7.62 ± 6.64	6.49 ± 6.02
Cold-start	6.62 ± 5.99	6.35 ± 5.50	8.22 ± 6.15	5.06 ± 5.55	7.98 ± 6.83	6.50 ± 6.02

on 245 examples and is then used to predict the ranking over all 72 vectors for the left out example in two variants.

In the first variant (the "reference" situation [C1]), a method is trained on data where the vector $\boldsymbol{y}_j$ *is part* of the *training set*, whereas in the second variant (the "cold start" situation [C2]) the same method is trained on data where $\boldsymbol{y}_j$ is completely *omitted*. In addition to the Kendall τ value that is used to quantify the quality of a predicted ranking in relation to a ground truth ranking, the deviation between the predicted rank of $\boldsymbol{y}_j$ and the true rank is recorded.

The empirical results given in Table 7.5 suggest that the Kendall τ performances of the reference and cold-start situations are almost identical. Therefore, side-information can be used beneficially with the BilinPL model.

7.3 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 rhythmic as well as timbre features were extracted and used as a feature representation.

To apply dyad ranking on this kind of multi-label data, the preference $(\boldsymbol{x}, y) \succ (\boldsymbol{x}, y')$ is constructed for each song $\boldsymbol{x}$ and each pair of emotions y, y' in such a way 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 a ranking represents a total 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, the 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 scaled to the unit interval, 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 Figure 7.7.

As can be seen, the unfolding nicely reflects the similarity between emotions. For example, *quiet-still* is located much closer to *relaxing-calm* than to *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.

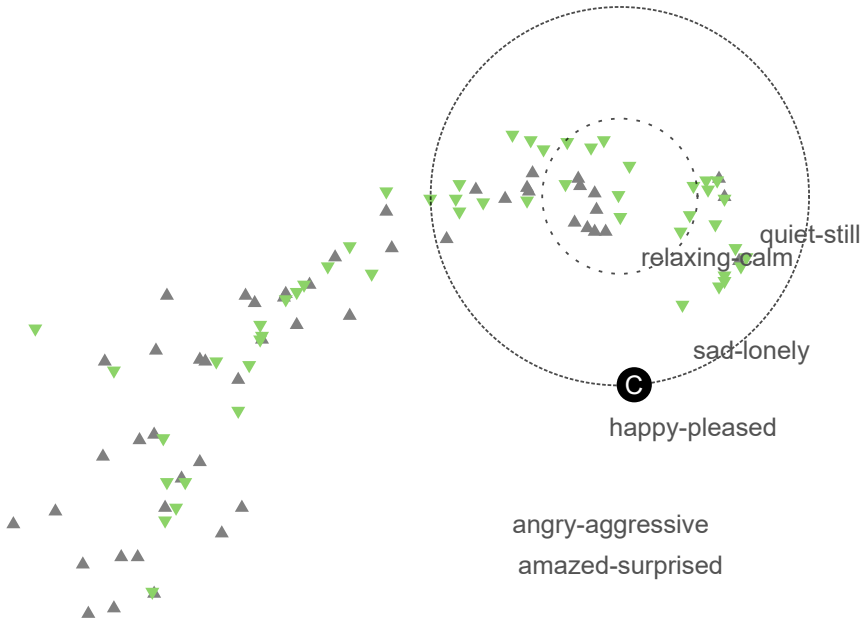

Figure 7.7: Unfolding representation of the modeled multi-label rankings of the Emotions dataset with PLNet. Triangles denote instances (upright/gray: training, upside-down/green: testing) and six class labels. For a test instance located at the center of the two dotted circles, a special calibration label (C) divides the set of labels into two groups. Labels closer to the test instance are more appropriate than those that are more distant.

7.4 Configuration of Image Processing Pipelines

The next case study elaborates on the idea of using PBRL for the purpose of algorithm selection and configuration (Brazdil and Giraud-Carrier, 2018), especially in domains where the results produced by an algorithm might be difficult to assess numerically. As an example, we consider the problem of configuring image-processing pipelines to enhance the quality of an input image. The idea is that for a human, a comparison between two candidate pictures $\boldsymbol{x}, \boldsymbol{x}'$ is again easier than an absolute quality assessment (here, we mimic such a comparison by applying a similarity measure, defining preference for $\boldsymbol{x}$ in terms of proximity to some reference $\boldsymbol{x}^*$).

An image-processing pipeline is a sequence of possibly parameterized operators where each operator takes an image as input and produces an image as output. The quality of a pipeline is influenced by the choice of operator types, the number of operators, their order, and, of course, the parameterization. We consider the choice of an operator with certain parameters as an action, which is taken by a policy learned with API-DR. The approach is outlined in Listing 10 and slightly differs from the basic version of Section 6.3.1.

Note that with the judgements on the quality of the pipelines, the function in Line 15 extracts pairwise preferences on (state,action) pairs, and all these preference pairs are added to the training set $\mathcal{T}$. The policy model is trained in a supervised way on these preferences at the end of each round and can then be used for the next round for further improvement.

The problem is related to *algorithm configuration* and *algorithm selection* (Lindauer et al., 2015). In algorithm configuration, there are instance vectors and one algorithm to process these instances. The algorithm can be adjusted with a parameter configuration and a model is sought, with the latter determining which parameters to set optimally for a new instance vector. This problem is different from the pipeline scenario as there are *several* configurable operators (algorithms) involved. In this aspect, the pipeline configuration problem is similar to the algorithm *selection* problem. That problem is about learning a model that can suggest one algorithm from a portfolio of diverse algorithms that delivers the best performance on a given instance. A crucial difference with that and also with the algorithm configuration problem is, however, that in the pipeline configuration problem not all instances exist a priori, but the majority of instances are

Algorithm 10 Pipeline Policy Training Algorithm

Require: Input $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{x}_n^*)\}_{n=1}^N$, max pipeline length L

Initialize random policy model π
repeat
  Sample a number of training examples $\mathcal{S} \subset \mathcal{D}$
  $\mathcal{T} = \emptyset$
  for $n = 1$ **to** $|\mathcal{S}|$ **do**
    for $l = 1$ **to** L **do**
      if $l = 1$ **then** $\boldsymbol{x}_n^{(l)} = \boldsymbol{x}_n$
      else $\boldsymbol{x}_n^{(l)} = \boldsymbol{x}'^{(l-1)}_n$
      end if
      for $i = 1$ **to** $|A|$ **do**
        $\boldsymbol{x}'_{n_i} = \text{apply operator}(\boldsymbol{x}_n^{(l)}, \boldsymbol{a}_i)$
        $\hat{\boldsymbol{x}}_{n_i} = \text{rollout}(\boldsymbol{x}'_{n_i}, \pi)$
      end for
      $\rho_n = \text{evaluate pipeline outputs } \{\hat{\boldsymbol{x}}_{n_i}\}_{i=1}^{|A|}$ ▷ human or machine (with $\boldsymbol{x}_n^*$)
      $\mathcal{T}_n = \text{generate pairwise preferences}(\rho_n)$
      $\mathcal{T} = \mathcal{T} \cup \mathcal{T}_n$
      $\boldsymbol{x}'^{(l)}_n = \text{choose subsequent state of the best performing pipeline } (\rho_n)$
    end for
  end for
  Train $(\pi, \mathcal{T})$
  Evaluate policy $(\pi, \mathcal{D})$
until No policy improvements
return π

created dynamically during the transformation process. Furthermore, the training signal is not available immediately after each configuration/algorithm choice, but only at the end of a pipeline evaluation.

7.4.1 Experimental Protocol

The policy model in this scenario is PLNet, which is capable of learning non-linear relationships between the preferences of state-operator configuration pairs $(\boldsymbol{x}, \boldsymbol{y})$. The input consists of a 1-of-K encoding for the pipeline operator positions and another 1-of-K encoding for the operator-parameter combination. Furthermore, PLNet is configured

with three layers, including one hidden layer with 10 neurons. All weights of the network are initialized randomly (by means of the uniform distribution) between -0.1 and 0.1, and the actual training is performed via stochastic gradient descent using 20 epochs and an initial learning rate of 0.1.

The set of image operators the learner can choose from consists of the logarithmic operator (Gonzalez and Woods, 2002), the γ operator, and the brightness operator. Each of those can be parameterized with different values (real numbers). Additionally, three further operators are available, namely an unsharping mask filter, histogram normalization, and a stop operator, all of which have no parameters. The stop operator enables a policy to control the length of a pipeline; it is usually applied when the outputs are good enough.

The images that are processed with the pipeline stem from the Fashion-MNIST dataset (Xiao et al., 2017). It consists of 60k training and 10k gray scale images, where each image has 28x28 pixels and belongs to one of 10 classes. The first hundred images from the original training set are used to create a pipeline training dataset that consists of distorted and ground truth image pairs. A distorted image, $\boldsymbol{x}$, is generated from a ground truth image by applying the pipeline $Op_1(2.5) \rightarrow Op_2(1.4) \rightarrow Op_1(1.5) \rightarrow Op_1(2.0)$ in reverse order on ground truth images $\boldsymbol{x}^*$. This essentially serves the purpose to examine whether or not the learner is able to recover the ground truth. A test dataset is generated in the same way on the first hundred images of the original test dataset.

As for the evaluation, we make use of the structured similarity (SSIM) measure (Wang et al., 2004). The overall quality of the policy model is measured in terms of the mean average error (MAE) between the produced and the ground truth images

$$E = \frac{1}{N} \sum_{n=1}^{N} |1 - sim(\boldsymbol{x}_n, \boldsymbol{x}_n^*)| \quad , \tag{7.1}$$

where N is the number of (distorted, ground truth) image pairs.

7.4.2 Results

The (averaged) learning curve of the learned policies is shown in Figure 7.8. It reflects the reduction in the error with an increasing number of rounds. The learning algorithm first enters an exploration phase, taking advantage of the (Boltzmann) exploration strategy of PBRL-DR, as described in Section 6.3.1.1. The latter is also responsible for the cool-down phase and the convergence of the policy.

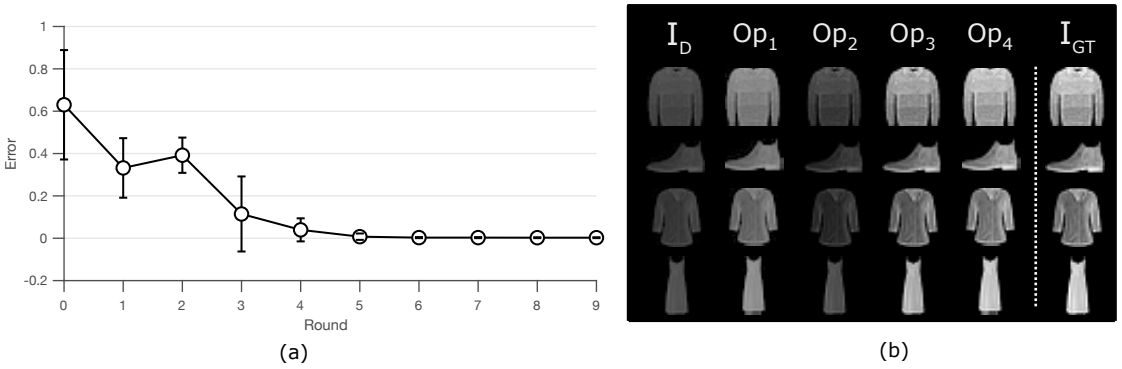

Figure 7.8: (a) Learning curve of the policy model over a number of rounds. (b) Image-processing pipeline with intermediate results. I_D refers to a damaged input and I_{GT} refers to the ground truth image. The output is provided after the application of the image operator Op_4.

7.5 Similarity Learning on Tagged Images

In this last case study, we apply dyad ranking for *similarity learning* on image data and make use of the DyMDS technique to visualize 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 dataset, 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. The study serves as a proof of concept for the case in dyad ranking where the data stems from a single common domain.

7.5.1 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 similarity learning, the reasonable assumption is made that a pair of images sharing the same label are mutually more similar to each other than a pair of images with different labels.

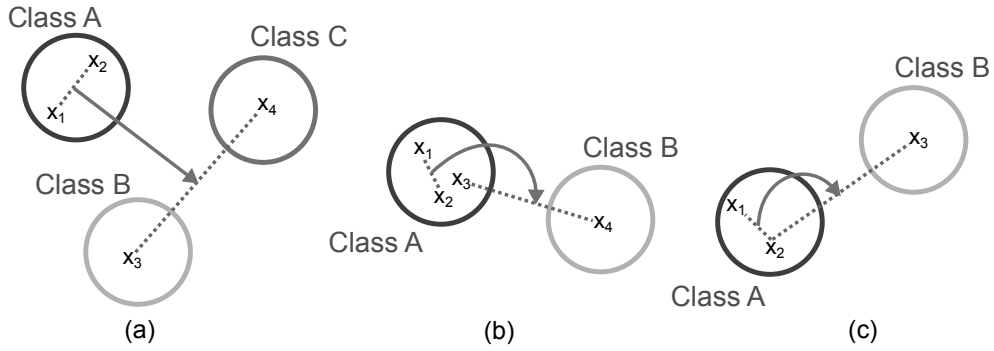

Figure 7.9: Elicitation of dyad rankings from multi-class data. Panels (a) to (c) show different ways to find dyad rankings of Length 2 in which the dyad at the first rank position contains instances that are more similar to each other compared to the instances contained in the dyad put at the second rank position.

Given a finite set of class labels, there are multiple ways to construct dyad rankings based on this idea, as depicted in Figure 7.9. In Panels (a)-(c), there are dyad rankings where 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 $(\boldsymbol{x}_1, \boldsymbol{x}_2) \succ (\boldsymbol{x}_3, \boldsymbol{x}_4)$. 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 the following contextualized preference statement: "Instance $\boldsymbol{x}_1$ is more similar to the 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 dataset 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. Section 4.3.3.5). 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. Listing 11), 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 speeds up the learning process because an equivalent model can be constructed with fewer parameter updates.

Algorithm 11 Selective Sampling with Bilinear PL

```
1: procedure SELECTIVESAMPLING(S, w)
2:     [x', y'] ← sampleDyad(S)
3:     repeat
4:         [x'', y''] ← sampleDyad(S)        ▷ See constraints (a)-(c) in Figure 7.9.
5:         x ← [x' ⊗ y' − x'' ⊗ y'']
6:     until (uncertaintyHigh(w, x)))
7:     return (x', y') ≻ (x'', y'')
8: end procedure
```

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 typically use training examples of Type (c) in Figure 7.9 (Bellet et al., 2013). To this end, of course, a suitable feature representation for the images is needed.

7.5.2 Construction of Image Features

The Caltech256 dataset 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 utilized 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 dataset (Jia et al., 2014; Krizhevsky et al., 2012). For each image, 4096 dimensional sparse feature vectors were obtained from the outputs of the sixth fully connected (6fc) layer of the convolutional neural network.

7.5.3 Application of DyMDS on Learned Image Similarity

The original data Caltech 256 dataset consists of 30607 images (in jpeg format) where each image belongs to one of the 256 classes (Griffin et al., 2007). We used a special subset of the dataset that comprises images from 10 different classes, as proposed in Appendix B of (Chechik et al., 2010). These are bear, skyscraper, billiards, yo-yo, minotaur, roulette-wheel, hamburger, laptop- 101, hummingbird, and blimp.

SiDRa was run with the learning rate $\eta = 0.1$ and the 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 Figure 7.9. The final BilinPL model comprised a weight matrix of 16 million elements ($= 4096^2$) and took 30K iterations.

The model was then used to calculate the matrix V of latent utility values for images in the test set. To apply the matrix on DyMDS, the matrix V is first made symmetrically and then the rank-transform (t_3) of values from the strictly lower-triangular matrix was performed. The upper triangular matrix is complemented by mirroring the values from the lower triangular matrix afterward.

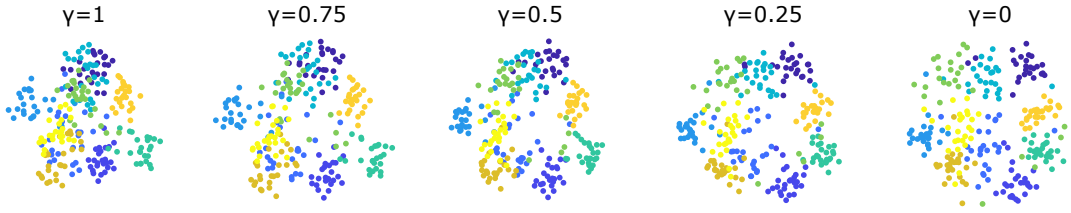

Figure 7.10: DyMDS configurations for varying γ values on the Caltech256 dataset.

The configurations produced by DyMDS for a varying number of γ values are shown in Figure 7.10. The different colors represent class labels, and it can be seen that points are clustered together which are from the same class. This confirms in principle the viability of dyad ranking as an approach toward similarity-learning.

Which of the thus produced configurations represents a good configuration cannot be answered in an unequivocal way. The reason for this is that a configuration can be considered in conjunction with ground-truth labels[2] as a kind of clustering solution. And clustering in principle is well-known to be an ill-posed problem (Jain, 2010). Therefore, it is hard to determine a good configuration.

Despite this difficulty, there exist measures that aim to characterize the quality of clustering. Internal clustering validity measures pick up properties that are intrinsic to datasets, and their mathematical formulation represents ideas of how good clusters should ideally be (Van Craenendonck and Blockeel, 2015). The following two measures are based on the idea that a good clustering can be expressed as the ratio of cluster compactness and separation. Compactness expresses that points within a cluster should be similar, while separation refers to the idea that points from different clusters should be dissimilar. The first of such measures is Davies–Bouldin (DB) (Davies and Bouldin, 1979). The compactness in DB is defined as the distance of points to its cluster centroid. And the separation is defined as the distances between all the cluster centroids. This measure should ideally be low. The second of those measures is Caliński-Harabasz (CH) (Caliński and Harabasz, 1974). The definition of compactness in CH coincides with that of DB. However, the definition for cluster separation is different, and it is based on the distances between cluster centroids and the global data centroid. In contrast to DB, this measure should ideally be high.

Both DB and CH are used to characterize the configurations on varying γ values, which are provided in Table 7.6. While a mixture of configurations with a γ value of 0.25 is preferred by the DB measure, a mixture with $\gamma = 0.5$ provides higher values of the CH measure. The latter configuration (with $\gamma = 0.5$) is chosen to create a visualization with the original images plotted on top of the configuration data points in Figure 7.11. Besides the grouping of similar objects into clusters, it seems that at a broader scale

[2]Ground-truth labels in Caltech256 are available because the dataset is typically used to apply multi-class methods on it.

Table 7.6: Cluster validity measures Davies–Bouldin and Caliński-Harabasz for DyMDS solutions with varying γ. The best results are indicated in bold-face.

γ	1.0	0.75	0.5	0.25	0.0
DB	2.021	1.519	1.302	**1.230**	1.267
CH	146.626	168.167	**171.377**	148.321	112.414

things with corners are organized on the right side, whereas more roundish things are located on the left side.

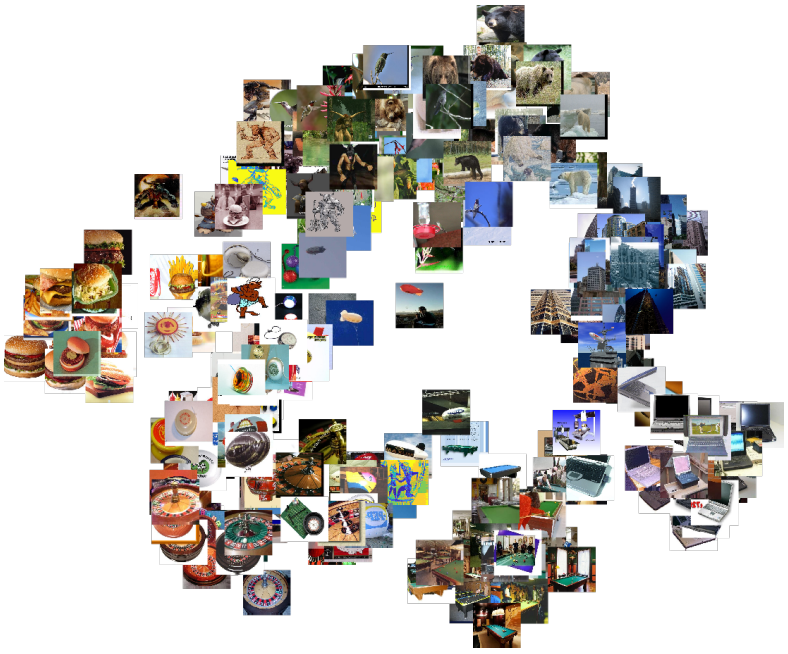

Figure 7.11: Dyadic MDS solution of Caltech256 Images with $\gamma = 0.5$.

8 Conclusion

The thesis addressed the research question of how preference learning on dyad ranking data can be tackled in a principled way.

Dyad ranking was introduced as a new preference learning setting that enables the unification of existing problems such as label and object ranking. It was defined formally in Chapter 2 and several prediction tasks were described henceforth. The setting extends label ranking in the sense that feature representations about labels can be taken into account. And it can also be considered as an extension of the object ranking setting in the sense that rankings of objects can be expressed under varying contexts. In comparison with label ranking, it was found that with dyad ranking (i) predictions become more reliable under the condition of scarce training information with the BilinPL model and (ii) ranking predictions become possible over labels that were not existent at the training phase, if additional descriptions about labels are utilized.

All dyad ranking models described in Chapter 4 have in common that they are generalizations of the PL model. The assumption underlying the JFPL model is that dyad rankings can be modeled with a PL model whose parameters are expressed as the functions of a weight vector and a joint-feature mapping representing the dyads. The BilinPL model was then introduced as a particular instantiation of it that imposes the Kronecker product as a concrete prescription for the joint-feature mapping function. Both the JFPL and the BilinPL model have in common that they require the proper engineering of feature vectors to capture preferences faithfully. In terms of the former model, these are single joint-feature vectors, while it requires the engineering of feature vector pairs for the latter. PLNet, in contrast, generalizes the PL model by being based on a neural network formulation of the PL model parameters. It differs from the former two in that joint-feature representations are learned instead of being engineered. It can furthermore capture non-linear relationships between input features and rankings.

By being based on the Plackett–Luce model, all introduced models have a probabilistic foundation. The benefits of possessing this key property could be demonstrated in terms of improving the learning process and enhancing preference visualizations with additional information concerning the uncertainty of the predictions.

In Chapter 5, the benefits of the latter aspect could be shown in an application where multidimensional unfolding was combined with dyad ranking models to provide visualizations. In Section 7.2.2, the performances of genetic algorithms on various instances of the traveling salesman problem modeled by the BilinPL model are visualized in an intuitive way. From an "ideal point" that represents a TSP problem, those genetic algorithms can be identified immediately for solving the problem well. The visualization is also used in Sections 7.1.3 and 7.3; this time in combination with PLNet. There it has been used to visualize the rankings of emotions for different pieces of music.

The case in dyad ranking in which dyads are composed of pairs from the *common* domain $\mathbb{X}$ is addressed with the application of dyad ranking for similarity learning in Section 7.5. A strong point of the approach is that the similarity obtained with the dyad ranking model SiDRa can again be used for the visualization with DyMDS from Section 5.4. This, in turn, makes it possible to indicate the uncertainty of the data similarity visually.

In Chapter 6, dyad ranking was utilized for preference-based reinforcement learning. To this end, API-DR was introduced as a novel approach for approximate policy iteration on dyadic preference data. It has the advantage of utilizing descriptions of actions and uses probabilistic information for balancing exploration and exploitation. An application of it is the learning of image-processing pipeline configurations, described in Section 7.4.

So far, the proposed models are learned with the maximum likelihood principle. A future work could be devoted to minimize other error measures, e.g. to emphasize top rank positions of the ranking predictions. This aspect is important if the number of objects to be ranked is high and if only the top-N ranked objects are of importance. This requirement is usually the case in large-scale information retrieval applications.

Future work could also be devoted to a generalization of the dyad ranking setting to support the ranking of k-tuples of feature vectors instead of dyads. As an example, dyad ranking would be limited in modeling the case where users (domain $\mathbb{X}$), items (domain $\mathbb{Y}$), and at the same time contexts as a third domain are subject to rankings.

Last but not least, the aspect of modeling ties in rankings has not received much attention in this work so far. And approaches beyond those that are based on the PL models are also conceivable for dyad ranking in future research.

9 Appendix

Definitions

Identity Matrix The identity matrix I_k is a $k \times k$ square diagonal matrix, in which the off-diagonal elements are all zero and the diagonal elements are all one. For example:

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Frobenius Norm

Let $\boldsymbol{U} \in \mathbb{R}^{N \times M}$ be a real-valued matrix with N rows and M columns. The *squared* Frobenius norm of matrix $\boldsymbol{U}$ is given by $\|\boldsymbol{U}\|_F^2 = \operatorname{tr} \boldsymbol{U}\boldsymbol{U}^\top = \sum_{i=1}^{N} \sum_{j=1}^{M} |u_{ij}|^2$.

Linear Map

Let V and W be vector spaces over $\mathbb{R}$. The function f is a linear map (or linear), if for any two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ and any scalar α the following two conditions are fulfilled:

$$\begin{aligned} f(\boldsymbol{x} + \boldsymbol{y}) &= f(\boldsymbol{x}) + f(\boldsymbol{y}) \\ f(\alpha \boldsymbol{x}) &= \alpha f(\boldsymbol{x}) \ . \end{aligned} \tag{9.1}$$

Scalar and Vector Projection

Let $\boldsymbol{v}$ and $\boldsymbol{w}$ be two vectors of a common vector space. The scalar projection of $\boldsymbol{v}$ onto $\boldsymbol{w}$ is given by $S_{\boldsymbol{w}}(\boldsymbol{v}) = \langle \boldsymbol{v}, \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \rangle \in \mathbb{R}$. And the vector projection of $\boldsymbol{v}$ onto $\boldsymbol{w}$ is a new vector in the direction of $\boldsymbol{w}$ defined as

$$\operatorname{Proj}_{\boldsymbol{w}}(\boldsymbol{v}) = S_{\boldsymbol{w}}(\boldsymbol{v}) \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} = \frac{\langle \boldsymbol{v}, \boldsymbol{w} \rangle}{\|\boldsymbol{w}\|^2} \boldsymbol{w} \ .$$

Iverson Bracket The Iverson bracket converts the logical proposition P into a number and is denoted by

$$[P] = \mathbf{1}_{\{P\}} = \begin{cases} 1 & \text{if P is true} \\ 0 & \text{if P is false} \end{cases} .$$

Vector Representation of a Matrix

Given a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ with

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} .$$

Then $\mathbf{vec}(\boldsymbol{X})$ is a vector representation of $\boldsymbol{X}$, in which the rows are stacked along one dimension, i.e.

$$\mathbf{vec}(\boldsymbol{X}) = (x_{11}, x_{12}, \dots, x_{1n}, x_{21}, x_{22} \dots, x_{mn}) = \boldsymbol{x} .$$

With the knowledge of m and n, the inverse operation $\mathbf{vec}^{-1}(\boldsymbol{x})$ returns the matrix $\boldsymbol{X}$ for the given vector $\boldsymbol{x}$.

Partial and Total Orders

The following terminology from *order theory* is often used in the label ranking literature to define basic concepts (Vembu and Gärtner, 2010). This happens, in particular, given the binary relation $\succ$ and the (finite) set $\mathcal{S}$. Given any elements $a, b, c \in \mathcal{S}$, we can consider the following properties:

(i) $a \succ a$ (reflexivity)

(ii) if $a \succ b$ and $b \succ a$ then $a = b$ (antisymmetry/asymmetry)

(iii) if $a \succ b$ and $b \succ c$ then $a \succ c$ (transitivity)

(iv) $a \succ b$ or $b \succ a$ (totality/comparability)

A binary relation, $\succ$, on $\mathcal{S}$ with Axioms (i)-(iii) is a partial order. The pair $(\mathcal{S}, \succ)$ is then called a *partially ordered set* or just *poset*. The partially ordered set $(\mathcal{S}, \succ)$ with Axioms (ii)-(iv) is called a totally ordered set. The relation $\succ$ with properties (ii)-(iv) is consequently called a *total order*. There are also other names for the total order relation;

these are: *linear order*, a *(strict) ranking*, or a *permutation*. In the case where ties exist, we speak of a *partial ranking*.

An alternative term for a ranking is a *(rank) ordering*, which is a sequence of elements where the element put on top has Rank 1, put second has Rank 2, and so on. An incomplete ordering is an ordering on a subset of available elements.

Kendall's Tau Rank Correlation Measure

The Kendall's tau coefficient (Kendall, 1938) is a rank correlation measure commonly used in the label ranking literature (Vembu and Gärtner, 2010; Zhou et al., 2014). It is defined as

$$\tau = \tau(\pi, \hat{\pi}) = \frac{C - D}{C + D} \tag{9.2}$$

$$= \frac{C - D}{K(K-1)/2} \tag{9.3}$$

$$= 1 - \frac{2D}{\binom{K}{2}} = 1 - \frac{4D}{K(K-1)} \tag{9.4}$$

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 assumes 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.

The importance of Kendall's τ for information retrieval has been shown in (Joachims, 2002). It is related to further ranking performance measures such as *Average Precision* and to *average rank* in case of a binary relevance scale.

Lemma 1. *Kendall's τ is a metric. For any $\pi, \sigma, \rho \in \mathbb{S}_M$ the following properties hold:*

(i) $\tau(\pi, \sigma) \geq 0$ *(non-negativity)*

(ii) $\tau(\pi, \sigma) = 0 \Leftrightarrow \pi = \sigma$ *(identity of indiscernibles)*

(iii) $\tau(\pi, \sigma) = \tau(\sigma, \pi)$ *(symmetry)*

(iv) $\tau(\pi, \sigma) \leq \tau(\pi, \rho) + \tau(\rho, \sigma)$ *(triangle inequality)*

Spearman's Rank Correlation Measure

Given the two rankings π, π' of the length K, the Spearman rank correlation measures the strength of the association between the rankings as

$$\rho = 1 - \frac{6D(\pi, \pi')}{K(K^2 - 1)} \ ,$$

with the sum of squared rank distances

$$D(\pi, \pi') = \sum_{i=1}^{m} \left[\pi(i) - \pi'(i)\right]^2 \ .$$

The measure ρ is, thus, a linear transformation (normalization) of D.

Derivation of Logit Choice Probabilities

The derivation to get from the discrete choice probability integral (4.13) to the closed-form logit choice probability (4.14) is given in (Train, 2009). This is a remarkable result because it enables the expression of choice probabilities in closed-form.

From (4.13) we have

$$P_{ni} = \int_{t=-\infty}^{\infty} \Big(\prod_{j \neq i} \exp(-\exp(-(t + V_{ni} - V_{nj}))) \Big) \exp(-t) \exp(-\exp(-t)) dt \ ,$$

with $t = \epsilon_{ni}$. By using $V_{ni} - V_{ni} = 0$ we have

$$\begin{aligned} P_{ni} &= \int_{t=-\infty}^{\infty} \Big(\prod_{j} \exp(-\exp(-(t + V_{ni} - V_{nj}))) \Big) \exp(-t) dt \\ &= \int_{t=-\infty}^{\infty} \exp\Big(-\sum_{j} \exp(-(t + V_{ni} - V_{nj})) \Big) \exp(-t) dt \\ &= \int_{t=-\infty}^{\infty} \exp\Big(-\exp(-t) \sum_{j} \exp(-(V_{ni} - V_{nj})) \Big) \exp(-t) dt \end{aligned}$$

By defining $s = \exp(-t)$, s.t. $-\exp(-t)dt = ds$ we can rewrite P_{ni} as

$$\begin{aligned} P_{ni} &= \int_{s=\infty}^{0} \exp\Big(- s \sum_j \exp(-(V_{ni} - V_{nj}))\Big)(-ds) \\ &= \int_{s=0}^{\infty} \exp\Big(- s \sum_j \exp(-(V_{ni} - V_{nj}))\Big) ds \\ &= \frac{\exp(-t \sum_j \exp(-(V_{ni} - V_{nj})))}{-\sum_j \exp(-(V_{ni} - V_{nj}))} \Bigg|_0^{\infty} \\ &= \frac{1}{\sum_j \exp(-(V_{ni} - V_{nj}))} = \frac{\exp(V_{ni})}{\sum_j \exp(V_{nj})} \quad . \quad \square \end{aligned}$$

Pseudocodes

Evaluation of a Ranking Probability

The effort for calculating the Plackett–Luce model probability $\mathbf{P}(\pi|\boldsymbol{v}) = \prod_{i=1}^{M-1} \frac{v_{\pi(i)}}{\sum_{j=i}^{M} v_{\pi(j)}}$ for the ranking π of the length M is $\mathcal{O}(M)$, see Algorithm 12.

Algorithm 12 Evaluation of Plackett–Luce Model Probability $\mathbf{P}(\pi|\boldsymbol{v})$

Require: Permutation π, Plackett–Luce parameter $\boldsymbol{v} \in \mathbb{R}^M$

1: $p, i \leftarrow 1$
2: $s \leftarrow \boldsymbol{v}_{\pi(M)}$
3: **for** $i < M - 1$ **do**
4: $\quad s \leftarrow s + \boldsymbol{v}_{\pi(M-i)}$
5: $\quad p \leftarrow p \cdot \boldsymbol{v}_{\pi(M-i)} / s$
6: $\quad i \leftarrow i + 1$
7: **end for**
8: **return** p

Sampling from the Plackett–Luce model distribution

Listing 13 describes the process of sampling a ranking from the Plackett–Luce model distribution. It mimics the vase metaphor from Silverberg (Marden, 1995) by drawing colored balls from a vase at several stages. It proceeds until all M colors are assigned to a rank. Another possibility to sample from the PL distribution can be realized by taking random samples from the Gumbel (double exponential or extreme value type I)

distribution whose PDF is

$$g(\boldsymbol{x}|\mu,\beta) = \frac{1}{\beta}\exp\left(-\frac{(x-\mu)}{\beta}\right)\exp\left(-\exp\left(-\frac{x-\mu}{\beta}\right)\right) \ .$$

PL parameters defined as $v_i = \exp\left(\frac{\mu_i}{\beta}\right)$ with a fixed $\beta = 1$ correspond to the i-th item in a ranking induced by the Thurstonian model (Guiver and Snelson, 2009). This relationship enables the alternative sampling algorithm in Listing 14.

Algorithm 13 Sampling from the Plackett–Luce model distribution (Vase)

Require: Plackett-Luce parameter $\boldsymbol{v} \in \mathbb{R}^M$

$M \leftarrow \text{length}(\boldsymbol{v})$
$\boldsymbol{r} \leftarrow \boldsymbol{v}$ ▷ Initialize remaining skills.
`stage` $\leftarrow 1$
for `stage` $< M$ **do**
  $\boldsymbol{p} \leftarrow \boldsymbol{r} / \sum \boldsymbol{r}_i$ ▷ Update probabilities vector.
  id $\leftarrow$ draw ball from vase ($\boldsymbol{p}$)
  `ordering`(`stage`) = id
  $\boldsymbol{r}$(id)= 0 ▷ Update remaining skills.
  `stage` $\leftarrow$ `stage` + 1
end for
return `ordering`

Algorithm 14 Sampling from the Plackett–Luce model distribution (Extreme Value)

Require: Plackett-Luce parameter $\boldsymbol{v} \in \mathbb{R}^M$

$\boldsymbol{r} \leftarrow$ take random sample from EV($\log(\boldsymbol{v})$, $\mathbf{1}$)
`ordering` = arg sort($\boldsymbol{r}$, descending)
return `ordering`

List of Figures

List of Tables

List of Algorithms

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283. 2016.

Adomavicius, G. and Tuzhilin, A. Context-aware recommender systems. In *Recommender systems handbook*, pages 191–226. Springer, 2015.

Agresti, A. *Categorical data analysis*. John Wiley & Sons, 2003.

Akata, Z., Perronnin, F., Harchaoui, Z., and Schmid, C. Label-embedding for attribute-based classification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 819–826. IEEE, 2013.

Akrour, R., Schoenauer, M., and Sebag, M. Preference-based policy learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6911, pages 12–27. 2011.

Aldrich, J. et al. Ra fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12(3):162–176, 1997.

Allison, P. D. and Christakis, N. A. Logit models for sets of ranked items. *Sociological methodology*, 24(1994):199–228, 1994.

Alvo, M. and Yu, P. L. *Statistical methods for ranking data*. Springer-Verlag New York, 2014.

Asuncion, A. and Newman, D. UCI machine learning repository. 2007.

Baeza-Yates, R. A. and Ribeiro-Neto, B. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

Bakir, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., and Vishwanathan, S. *Predicting Structured Data*. Advances in neural information processing systems. MIT Press, Cambridge, MA, USA, 2007.

Basilico, J. and Hofmann, T. Unifying collaborative and content-based filtering. In *Proceedings of the 21st International Conference on Machine Learning*, page 9. ACM Press, New York, New York, USA, 2004.

Bayer, I., He, X., Kanagal, B., and Rendle, S. A generic coordinate descent framework for learning from implicit feedback. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1341–1350. International World Wide Web Conferences Steering Committee, 2017.

Bell, S. and Bala, K. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 34(4):98, 2015.

Bellet, A., Habrard, A., and Sebban, M. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, page 57, 2013.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.

Bishop, C. M. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

Borg, I. *Anwendungsorientierte Multidimensionale Skalierung*. Springer-Verlag, 1981.

Borg, I. and Groenen, P. *Modern Multidimensional Scaling - Theory and Applications*. Springer, 2005.

Borg, I., Groenen, P. J., and Mair, P. *Applied Multidimensional Scaling*. Springer Publishing Company, Incorporated, 2012.

Bottou, L. Online algorithms and stochastic approximations. In D. Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. Revised, oct 2012.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187. Springer, Paris, France, 2010.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, Reading, MA, 2004.

Bradley, R. A. and Terry, M. E. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4), 1952.

Brazdil, P. and Giraud-Carrier, C. G. Metalearning and algorithm selection: progress, state of the art and introduction to the 2018 special issue. *Machine Learning*, 107(1):1–14, 2018.

Brazdil, P., Giraud-Carrier, C., Soares, C., and Vilalta, R. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2008.

Breese, J. S., Heckerman, D., and Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

Brin, S. and Page, L. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

Brinker, K., Fürnkranz, J., and Hüllermeier, E. A unified model for multilabel classification and ranking. In *Proceedings of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29–September 1, 2006, Riva del Garda, Italy*, pages 489–493. IOS Press, 2006.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. Signature verification using a "siamese" time delay neural network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 737–744. Morgan Kaufmann Publishers Inc., 1993.

Burges, C. J. C. From RankNet to LambdaRank to LambdaMART: An Overview. Technical report, Microsoft Research, 2010.

Burges, C. J., Ragno, R., and Le, Q. V. Learning to rank with nonsmooth cost functions. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 193–200. MIT Press, 2007.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. Learning to rank using gradient descent. In *Proceedings ICML, 22nd International Conference on Machine Learning*, pages 89–96. ACM, New York, NY, USA, 2005.

Busing, F. *Advances in Multidimensional Unfolding*. Ph.D. thesis, University of Leiden, 2010.

Caliński, T. and Harabasz, J. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. Learning to rank: From pairwise approach to listwise approach. In *Proceedings ICML, 24th International Conference on Machine Learning*, pages 129–136. ACM, New York, NY, USA, 2007.

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

Carroll, J. D. and Chang, J.-H. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrica*, 35(3):238–319, 1970.

Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. Boltzmann exploration done right. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6284–6293. Curran Associates, Inc., 2017.

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

Chechik, G., Sharma, V., Shalit, U., and Bengio, S. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research*, 11:1–29, 2010. ISSN 1532-4435.

Chen, K., Li, R., Dou, Y., Liang, Z., and Lv, Q. Ranking support vector machine with kernel approximation. *Computational intelligence and neuroscience*, 2017, 2017.

Cheng, W., Dembczyński, K., and Hüllermeier, E. Label ranking methods based on the Plackett-Luce model. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 215–222. Omnipress, Haifa, Israel, 2010.

Cheng, W., Fürnkranz, J., Hüllermeier, E., and Park, S. H. Preference-based policy iteration: Leveraging preference learning for reinforcement learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6911 LNAI:312–327, 2011.

Cheng, W., Hühn, J., and Hüllermeier, E. Decision tree and instance-based learning for label ranking. In *Proceedings ICML, 26th International Conference on Machine Learning*, pages 161–168. Omnipress, Montreal, Canada, 2009.

Cheng, W. and Hüllermeier, E. Learning similarity functions from qualitative feedback. In *Proc. ECCBR–2008, 9th European Conference on Case-Based Reasoning*, number 5239 in LNAI, pages 120–134. Springer-Verlag, Trier, Germany, 2008.

Cheng, W. and Hüllermeier, E. Probability estimation for multi-class classification based on label ranking. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 83–98. Springer, 2012.

Cheng, W., Hüllermeier, E., Waegeman, W., and Welker, V. Label ranking with partial abstention based on thresholded probabilistic models. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2501–2509. Curran Associates, Inc., 2012.

Chopra, S., Hadsell, R., and LeCun, Y. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

Cohen, W. W., Schapire, R. E., and Singer, Y. Learning to order things. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 451–457. MIT Press, 1998.

Cohen, W., Schapire, R., and Singer, Y. Learning to order things. *Journal of Artificial Intelligence Research*, 10(1):243–270, 1999.

Coombs, C. H. Psychological scaling without a unit of measurement. *Psychological Review*, 57(3):145–158, 1950.

Cortes, C., Mohri, M., and Rastogi, A. Magnitude-preserving ranking algorithms. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 169–176. ACM, New York, NY, USA, 2007.

Critchlow, D. E. and Fligner, M. A. Ranking models with item covariates. In *Probability models and statistical analyses for ranking data*, pages 1–19. Springer, 1993.

David, H. A. *The method of paired comparisons*. Griffin, London, 2 edition, 1969.

Davies, D. L. and Bouldin, D. W. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

De Leeuw, J. Applications of convex analysis to multidimensional scaling. In *Recent Developments in Statistics*, pages 133–146. North Holland Publishing Company, 1977.

De Leeuw, J. and Heiser, W. J. Convergence of correction matrix algorithms for multidimensional scaling. *Geometric representations of relational data*, pages 735–752, 1977.

De Leeuw, J. and Mair, P. Multidimensional scaling using majorization: Smacof in r. *Journal of Statistical Software*, 31(1):1–30, 2009.

Dekel, O., Singer, Y., and Manning, C. D. Log-linear models for label ranking. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16 (NIPS-2003)*, pages 497–504. MIT Press, 2004.

Dempster, A., Laird, N., and Rubin, D. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

Diaconis, P. and Graham, R. L. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 262–268, 1977.

Dimitrakakis, C. and Lagoudakis, M. G. Rollout sampling approximate policy iteration. *Machine Learning*, 72(3):157–171, 2008.

Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification (2nd Edition)*. Wiley, 2000.

Engl, H., Hanke, M., and Neubauer, A. *Regularization of Inverse Problems*. Kluwer Academic Publishers, 1996.

Evgeniou, T., Pontil, M., and Poggio, T. Regularization networks and support vector machines. *Advances in computational mathematics*, 13(1):1, 2000.

Fei-Fei, L., Fergus, R., and Perona, P. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

Fisher, R. A. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222:309–368, 1922.

Fligner, M. A. and Verducci, J. S. Distance based ranking models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 359–369, 1986.

Freund, Y. and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

Frome, A., Corrado, G. S., Shlens, J., Bengio, S., Dean, J., Ranzato, M. A., and Mikolov, T. Devise: A deep visual-semantic embedding model. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2121–2129. Curran Associates, Inc., 2013.

Fürnkranz, J. and Hüllermeier, E. Pairwise preference learning and ranking. In N. Lavrač, D. Gamberger, H. Blockeel, and L. Todorovski, editors, *Proceedings of the 14th European Conference on Machine Learning (ECML-03)*, volume 2837 of *Lecture Notes in Artificial Intelligence*, pages 145–156. Springer-Verlag, Cavtat, Croatia, 2003.

Fürnkranz, J. and Hüllermeier, E. *Preference Learning*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

Fürnkranz, J. and Hüllermeier, E. Preference learning: An introduction. In J. Fürnkranz and E. Hüllermeier, editors, *Preference Learning*, pages 1–17. Springer-Verlag, 2010a.

Fürnkranz, J. and Hüllermeier, E. Preference learning and ranking by pairwise comparison. In J. Fürnkranz and E. Hüllermeier, editors, *Preference Learning*, pages 65–82. Springer-Verlag, 2010b.

Fürnkranz, J. and Hüllermeier, E. Preference learning. In C. Sammut and G. Webb, editors, *Encyclopedia of Machine Learning and Data Mining*. Springer, 2016.

Fürnkranz, J., Hüllermeier, E., Cheng, W., and Park, S.-H. Preference-based reinforcement learning: a formal framework and a policy iteration algorithm. *Machine Learning*, 89(1-2):123–156, 2012.

Fürnkranz, J., Hüllermeier, E., Loza Mencía, E., and Brinker, K. Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153, 2008.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323. PMLR, Fort Lauderdale, FL, USA, 2011.

Gonzalez, R. C. and Woods, R. E. *Digital Image Processing (2nd Ed)*. Prentice Hall, 2002.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016.

Gormley, I. C. and Murphy, T. B. Clustering ranked preference data using sociodemographic covariates. *Choice modelling: the state-of-the-art and the state-of-practice*, 2010.

Gower, J. C. Generalized procrustes analysis. *Psychometrika*, 40(1):33–51, 1975.

Griffin, G., Holub, a., and Perona, P. Caltech-256 object category dataset. *Caltech mimeo*, 11:20, 2007.

Groenen, P. J. and Heiser, W. J. The tunneling method for global optimization in multidimensional scaling. *Psychometrika*, 61(3):529–550, 1996.

Groenen, P. and van de Velden, M. Multidimensional scaling by majorization: A review. *Journal of Statistical Software*, 73(1):1–26, 2016.

Guiver, J. and Snelson, E. Bayesian inference for plackett-luce ranking models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 377–384. ACM, 2009.

Hamm, J. and Belkin, M. Probabilistic zero-shot classification with semantic rankings. *CoRR*, abs/1502.08039, 2015.

Han, X., Leung, T., Jia, Y., Sukthankar, R., and Berg, A. C. Matchnet: Unifying feature and metric learning for patch-based matching. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3279–3286. 2015.

Har-Peled, S., Roth, D., and Zimak, D. Constraint classification: A new approach to multiclass classification. In *Proceedings ALT, 13th International Conference on Algorithmic Learning Theory*, pages 365–379. Springer, 2002a.

Har-Peled, S., Roth, D., and Zimak, D. Constraint classification for multiclass classification and ranking. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 809–816. MIT Press, 2002b.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

Herbrich, R., Graepel, T., Bollmann-Sdorra, P., and Obermayer, K. Learning a preference relation for information retrieval. In *Working Notes of the AAAI Workshop on Information Retrieval 98*, pages 83 – 86. 1998.

Hoffman, S. D. and Duncan, G. J. Multinomial and conditional logit discrete-choice models in demography. *Demography*, 25(3):415–427, 1988.

Hofmann, T., Puzicha, J., and Jordan, M. I. Learning from dyadic data. In *Advances in neural information processing systems (NIPS-1998*, pages 466–472. 1999.

Hu, Y., Koren, Y., and Volinsky, C. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272. IEEE Computer Society, 2008.

Hüllermeier, E. and Fürnkranz, J. Comparison of ranking procedures in pairwise preference learning. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04)*. Perugia, Italy, 2004.

Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16):1897–1916, 2008.

Hunter, D. R. Mm algorithms for generalized bradley-terry models. *Annals of Statistics*, 32(1):384–406, 2004.

Huybrechts, G. *Learning to rank with deep neural networks.* Master's thesis, Ecole polytechnique de Louvain (EPL), 2016.

Jain, A. K. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

Järvelin, K. and Kekäläinen, J. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48. ACM, 2000.

Järvelin, K. and Kekäläinen, J. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Joachims, T. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

Joachims, T. and Radlinski, F. Search engines that learn from implicit feedback. *IEEE Computer*, 40(8):34–40, 2007.

Kamishima, T., Kazawa, H., and Akaho, S. Supervised ordering - an empirical survey. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 673–676. 2005.

Kamishima, T., Kazawa, H., and Akaho, S. A survey and empirical comparison of object ranking methods. In *Preference learning*, pages 181–201. Springer, 2011.

Kanda, J., Soares, C., Hruschka, E. R., and de Carvalho, A. C. P. L. F. 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*, pages 488–495. Springer, Doha, Qatar, 2012.

Kendall, M. G. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

Khardon, R. and Wachman, G. Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8:227–248, 2007.

Kidwell, P., Lebanon, G., and Cleveland, W. Visualizing incomplete and partially ranked data. *IEEE Transactions on visualization and computer graphics*, 14(6):1356–1363, 2008.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Koren, Y. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

Koren, Y. and Bell, R. Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer, 2015.

Krantz, D. H. Rational distance functions for multidimensional scaling. *Journal of Mathematical Psychology*, 4(2):226–245, 1967.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Kruskal, J. B. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

Kuleshov, V. and Doina, P. Algorithms for the multi-armed bandit problem. *Journal of Machine Learning*, 2010.

Lagoudakis, M. and Parr, R. Reinforcement learning as classification: Leveraging modern classifiers. *In Proceedings of the Twentieth International Conference on Machine Learning*, 20(1):424, 2003.

Lampert, C. H., Nickisch, H., and Harmeling, S. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, 2014.

Lange, K. *MM Optimization Algorithms*. SIAM-Society for Industrial and Applied Mathematics, 2016.

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

Larochelle, H., Erhan, D., and Bengio, Y. Zero-data learning of new tasks. In *AAAI*, volume 1, page 3. 2008.

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

Lazaric, A. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

Lee, C.-W., Fang, W., Yeh, C.-K., and Wang, Y.-C. F. Multi-label zero-shot learning with structured knowledge graphs. *arXiv preprint arXiv:1711.06526*, 2017.

Lewis, D. D. and Gale, W. A. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 3–12. Springer-Verlag New York, Inc., New York, NY, USA, 1994.

Lichman, M. UCI machine learning repository. 2013.

Lindauer, M., Hoos, H. H., Hutter, F., and Schaub, T. Autofolio: Algorithm configuration for algorithm selection. In *AAAI Workshop: Algorithm Configuration*. 2015.

Littlestone, N. and Warmuth, M. The weighted majority algorithm. *Information and Computation*, 108(2):212 – 261, 1994.

Liu, T.-Y. *Learning to rank for information retrieval*. Springer, 2011.

Liu, D. C. and Nocedal, J. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Luce, R. D. *Individual Choice Behavior: A theoretical analysis*. Wiley, 1959.

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

Luce, R. D. and Suppes, P. Preference, utility, and subjective probability. *Handbook of mathematical psychology*, 3:249–410, 1965.

Luenberger, D. G. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, 1973.

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

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

Marden, J. I. *Analyzing and Modeling Rank Data*. Chapman & Hall, 1995.

Marschak, J. Binary choice constraints on random utility indicators. Cowles Foundation Discussion Papers 74, Cowles Foundation for Research in Economics, Yale University, 1959.

Maystre, L. and Grossglauser, M. Fast and accurate inference of plackett–luce models. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 172–180. Curran Associates, Inc., 2015.

McFadden, D. Conditional logit analysis of qualitative choice behavior. *Frontiers in Econometrics*, pages 105–142, 1973.

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

Menon, A. K. *Latent feature models for dyadic prediction*. Ph.D. thesis, University of California, San Diego, 2013.

Menon, A. K. and Elkan, C. Dyadic prediction using a latent feature log-linear model. *arXiv preprint arXiv:1006.2156*, 2010a.

Menon, A. K. and Elkan, C. A log-linear model with latent features for dyadic prediction. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 364–373. IEEE Computer Society, 2010b.

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

Mensink, T., Verbeek, J., Perronnin, F., and Csurka, G. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *Computer Vision–ECCV 2012*, pages 488–501. Springer, 2012.

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

Misir, M. and Sebag, M. Algorithm selection as a collaborative filtering problem. Research report, INRIA, 2013.

Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning*. The MIT Press, 2012.

Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

Nair, V. and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814. Omnipress, 2010.

Nilsson, N. J. *Learning Machines: Foundations of trainable pattern-classifying systems*. McGraw-Hill, New York, NY, USA, 1965.

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

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

Pahikkala, T., Stock, M., Airola, A., Aittokallio, T., De Baets, B., and Waegeman, W. A two-step learning approach for solving full and almost full cold start problems in dyadic prediction. In T. Calders, F. Esposito, E. Hüllermeier, and R. Meo, editors, *Proceedings ECML/PKDD–2014, European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 517–532. Springer Berlin Heidelberg, Nancy, France, 2014.

Pahikkala, T., Tsivtsivadze, E., Airola, A., Boberg, J., and Salakoski, T. Learning to rank with pairwise regularized least-squares. In *SIGIR 2007 workshop on learning to rank for information retrieval*, pages 27–33. 2007.

Pahikkala, T., Waegeman, W., Airola, A., Salakoski, T., and De Baets, B. Conditional ranking on relational data. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part II*, pages 499–514. Springer Berlin Heidelberg, 2010.

Palatucci, M., Pomerleau, D., Hinton, G. E., and Mitchell, T. M. Zero-shot learning with semantic output codes. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1410–1418. Curran Associates, Inc., 2009.

Park, Y. and Marcotte, E. M. Flaws in evaluation schemes for pair-input computational predictions. *Nature methods*, 9(12):1134, 2012.

Patterson, D. W. *Artificial Neural Networks: Theory and Applications*. Prentice Hall PTR, 1st edition, 1998.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. G. Meta-learning by landmarking various learning algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 743–750. Morgan Kaufmann Publishers Inc., 2000.

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

Pleskac, T. J. Decision and choice: Luce's choice axiom. In P. Bona, editor, *International encyclopedia of social and behavioral sciences (2nd ed.* Department of Psychology, Michigan State University, 2013.

Prechelt, L. Early stopping—but when? In *Neural Networks: Tricks of the Trade*, pages 53–67. Springer, 2012.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

Ribeiro, G., Duivesteijn, W., Soares, C., and Knobbe, A. J. Multilayer perceptron for label ranking. In *Proceedings ICANN, 22nd International Conference on Artificial Neural Networks*, pages 25–32. Springer, Lausanne, Switzerland, 2012.

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

Robertson, S., Zaragoza, H., et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

Rojas, R. *Theorie der neuronalen Netze: Eine systematische Einführung.* Springer, 1993.

Romera-Paredes, B. and Torr, P. An embarrassingly simple approach to zero-shot learning. In *International Conference on Machine Learning*, pages 2152–2161. 2015.

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

Salakhutdinov, R. and Mnih, A. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20. 2008.

Sappadla, P. V., Nam, J., Loza Mencía, E., and Fürnkranz, J. Using semantic similarity for multi-label zero-shot classification of text documents. In *Proceedings of the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN-16)*. d-side publications, Bruges, Belgium, 2016.

Schäfer, D. A latent-feature Plackett-Luce model for dyad ranking completion. In *Workshop KDML@LWDA, "Learning, Knowledge, Data, Analytics"*. Rostock, Germany, 2017.

Schäfer, D. and Hüllermeier, E. Dyad ranking using a bilinear Plackett-Luce model. In *Proceedings ECML/PKDD–2015, European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 227–242. Springer, Porto, Portugal, 2015a.

Schäfer, D. and Hüllermeier, E. Preference-based meta-learning using dyad ranking: Recommending algorithms in cold-start situations. In J. Vanschoren, P. Brazdil, C. G. Giraud-Carrier, and L. Kotthoff, editors, *MetaSel@PKDD/ECML*, volume 1455 of *CEUR Workshop Proceedings*, pages 110–111. CEUR-WS.org, 2015b.

Schäfer, D. and Hüllermeier, E. Plackett-Luce networks for dyad ranking. In *Workshop KDML@LWDA, "Learning, Knowledge, Data, Analytics"*. Potsdam, Germany, 2016a.

Schäfer, D. and Hüllermeier, E. Preference-based reinforcement learning using dyad ranking. In *EURO mini conference: DA2PL (From Multiple Criteria Decision Aid to Preference Learning)*. 2016b.

Schäfer, D. and Hüllermeier, E. Dyad ranking using plackett–luce models based on joint feature representations. *Machine Learning*, 2018.

Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

Sculley, D. Large scale learning to rank. In *In NIPS 2009 Workshop on Advances in Ranking*, pages 58–63. 2009.

Settles, B. Active learning literature survey. Technical Report 1648, University of Wisconsin-Madison, 2010.

Shalev-Shwartz, S. and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

Shi, Y., Larson, M., and Hanjalic, A. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)*, 47(1):3, 2014.

Spearman, C. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

Stern, D., Samulowitz, H., Pulina, L., and Genova, U. Collaborative expert portfolio management. *Artificial Intelligence*, 116(3):179–184, 2010.

Stevens, S. S. On the theory of scales of measurement. *Science*, 103(2684):677–680, 1946.

Stock, M., Pahikkala, T., Airola, A., De Baets, B., and Waegeman, W. A comparative study of pairwise learning methods based on kernel ridge regression. *Neural Computation*, 30(8):2245–2283, 2018.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

Sutton, R. and Barto, A. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

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

Thurstone, L. L. A law of comparative judgment. *Psychological review*, 34(4):273, 1927.

Train, K. E. *Discrete choice methods with simulation*. Cambridge university press, 2009.

Trohidis, K., Tsoumakas, G., Kalliris, G., and Vlahavas, I. P. Multi-label classification of music into emotions. In J. P. Bello, E. Chew, and D. Turnbull, editors, *Proceedings of the 9th International Conference on Music Information Retrieval*, pages 325–330. Philadelphia, USA, 2008.

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

Tucker, L. R. Intra-individual and inter-individual multidimensionality. 1960.

Van Craenendonck, T. and Blockeel, H. Using internal validity measures to compare clustering algorithms. In *Proceedings of the Workshop on Automatic Machine Learning, AutoML 2015, co-located with 32nd International Conference on Machine Learning (ICML 2015)*. 2015.

Van Deun, K., Groenen, P., and Delbeke, L. Vipscal: A combined vector ideal point model for preference data. Econometric institute report no. ei 2005-03, 2005.

Vanderbei, R. J. Loqo: An interior point code for quadratic programming. *Optimization methods and software*, 11(1-4):451–484, 1999.

Veit, A., Kovacs, B., Bell, S., McAuley, J., Bala, K., and Belongie, S. Learning visual clothing style with heterogeneous dyadic co-occurrences. In *International Conference on Computer Vision (ICCV)*. 2015.

Vembu, S. and Gärtner, T. Label ranking: a survey. In J. Fürnkranz and E. Hüllermeier, editors, *Preference Learning*. Springer, 2010.

Vembu, S. and Gärtner, T. Label ranking algorithms: A survey. In J. Fürnkranz and E. Hüllermeier, editors, *Preference Learning*, pages 45–64. Springer-Verlag, 2010.

Volkovs, M. and Zemel, R. S. Collaborative ranking with 17 parameters. In *Advances in Neural Information Processing Systems*, pages 2294–2302. 2012.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

Wang, L., Li, Y., Huang, J., and Lazebnik, S. Learning two-branch neural networks for image-text matching tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

Wang, J., Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., and Wu, Y. Learning fine-grained image similarity with deep ranking. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1386–1393. IEEE Computer Society, 2014.

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8(3):272–292, 1992.

Weimer, M., Karatzoglou, A., Le, Q. V., and Smola, A. J. Cofi rank - maximum margin matrix factorization for collaborative ranking. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1593–1600. MIT Press, Cambridge, MA, 2008.

Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph.D. thesis, Harvard University, Cambridge, MA, 1974.

Xia, F., Liu, T.-Y., Li, H., and Li, H. Top-k consistency of learning to rank methods. Technical report, 2009a.

Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. Listwise approach to learning to rank. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1192–1199, 2008.

Xia, F., yan Liu, T., and Li, H. Statistical consistency of top-k ranking. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 2098–2106. Curran Associates, Inc., 2009b.

Xian, Y., Schiele, B., and Akata, Z. Zero-shot learning-the good, the bad and the ugly. *arXiv preprint arXiv:1703.04394*, 2017.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. 2017.

Yellott Jr, J. I. The relationship between luce's choice axiom, thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, 15(2):109–144, 1977.

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