ECAI MONTPELLIER 2012

Workshop Proceedings

PREFERENCE LEARNING:

PROBLEMS AND APPLICATIONS IN AI

PL-12

August 28, 2012

Montpellier, France

Preference Learning: Problems and Applications in AI (PL-12)

The topic of *preferences* has recently attracted considerable attention in Artificial Intelligence (AI) research, notably in fields such as agents, non-monotonic reasoning, constraint satisfaction, planning, and qualitative decision theory. Preferences provide a means for specifying desires in a declarative way, which is a point of critical importance for AI. Drawing on past research on knowledge representation and reasoning, AI offers qualitative and symbolic methods for treating preferences that can reasonably complement hitherto existing approaches from other fields, such as decision theory and economic utility theory. Needless to say, however, the acquisition of preferences is not always an easy task. Therefore, not only are modeling languages and representation formalisms needed, but also methods for the automatic *learning*, *discovery*, and *adaptation* of preferences. It is hence hardly surprising that methods for learning and predicting preference models from explicit or implicit preference information and feedback are among the very recent research trends in machine learning and related areas.

The goal of this workshop is on the one hand to continue a series of successful workshops at the ECML/PKDD conference series (PL-08, PL-09, PL-10), but, more importantly, also to expand the scope by drawing the attention to a broader AI audience. In particular, we seek to figure out new problems and applications of preference learningnatural language processing, game playing, decision making and planning. Indeed, we believe that there is a strong potential for preference learning techniques in these areas, which has not yet been fully explored.

The papers presented in this workshop provide a spotlight on ongoing research in this area. They make technical contributions to the field, but also show work in progress in this area. We hope that these proceedings help to spark further interest in this emerging research area and contribute to establishing preference learning as an important in AI research.

Johannes Fürnkranz
Eyke Hüllermeier

Contents

A Preliminary Study on a Recommender System for the Million Songs Dataset Challenge

Fabio Aiolli[1]

Abstract. In this paper the preliminary study we are conducting on the Million Songs Dataset (MSD) challenge is described. The task of the competition is to suggest a set of songs to a user given half of its listening history and complete listening history of other 1 million people. We focus on memory-based collaborative filtering approaches since they are able to deal with large datasets in an efficient and effective way. In particular, we investigated on *i)* defining suitable similarity functions, *ii)* studying the effect of the locality of the collaborative scoring function, and *iii)* aggregating multiple ranking strategies to define the overall recommendation. Using these techniques we are in the top positions according to the current standing of the competition leaderboard (at the moment of this writing the challenge has about 150 registered teams).

1 Introduction

The Million Song Dataset Challenge [5] is a large scale, music recommendation challenge, where the task is to predict which songs a user will listen to, provided the listening history of the user. The challenge is based on the Million Song Dataset (MSD), a freely-available collection of meta data for one million of contemporary songs (e.g. song titles, artists, year of publication, audio features, and much more) [2]. About one hundred and fifty teams are currently participating to the challenge. The subset of data actually used in the challenge is the so called Taste Profile Subset that consists of more than 48 million triplets *(user,song,count)* gathered from user listening histories. Data consists of about 1.2 million users and covers more than 380,000 songs in MSD. The user-item matrix is very sparse as the fraction of non-zero entries (the density) is only 0.01%.

Collaborative Filtering (CF) is a technology which uses the items by user matrix to discover other users with similar tastes as the active user for which we want to make the prediction. The intuition is that if other users, similar to the active user, already purchased a certain item, then it is likely that the active user will like that item as well. A similar (dual) consideration can be made by changing the point of view. If we know that a set of items are often purchased together (they are similar in some sense), then, if the active user has bought one of them, probably he/she will be interested to the other as well. The first view is the one that is prevalent in recent CF literature. In this paper, we show that the second view turned out more useful when used for the MSD competition.

In Section 2 collaborative filtering is described and proposed as a first approach to solve the problem of MSD. In the same section three different views of the memory-based CF task are proposed that motivated the different algorithms of the section. In Section 3 empirical results of the proposed techniques are presented and discussed.

2 A CF approach to the MSD task

Collaborative filtering techniques use a database in the form of a user-item matrix R of preferences. In a typical CF scenario a set $\mathcal{U}$ of n users and a set $\mathcal{I}$ of m items exist and the entries of $R = \{r_{ui}\} \in \mathbb{R}^{n \times m}$ represent how much user u likes item i. In this paper, we assume $r_{ui} \in \{0, 1\}$ as this is the setting of the MSD challenge. Entries r_{ui} represent the fact that user u have listened to (or would like to listen to) the song i. In the following we refer to items or songs interchangeably. The MSD challenge task is more properly described as a *top-τ recommendation* task. Specifically, we want to identify a list of τ ($\tau = 500$ in the challenge) items $I_u \subseteq \mathcal{I}$ that *active user* u will like the most. Clearly, this set must be disjoint with the set of items already rated (purchased, or listened to) by the active user.

2.1 Memory-based CF

In memory-based CF algorithms the entire user-item matrix is used to generate a prediction. Generally, given a new user for which we want to obtain the prediction, the set of items to suggest are computed looking at similar users. This strategy is typically referred to as *user-based recommendation*. Alternatively, in the *item-based recommendation* strategy, one computes the most similar items for the items that have been already purchased by the active user, and then aggregates those items to form the final recommendation. There are many different proposal on how to aggregate the information provided by similar users/items (see [7] for a good survey). We focus on the simple weighted sum strategy. A deeper analysis of this simple strategy allows us to highlight an interesting duality that exists between user-based and item-based recommendation algorithms.

In the *user-based* type of recommendation, the scoring function, on the basis of which the recommendation is made, is computed by

$$h^U_{ui} = \sum_{v \in \mathcal{U}} f(w_{uv}) r_{vi} = \sum_{v \in \mathcal{U}(i)} f(w_{uv}),$$

that is, the score obtained on an item for a target user is proportional to the similarities between the target user u and other users v that have purchased the item i ($v \in \mathcal{U}(i)$). This score will be higher for items which are often rated by similar users.

On the other hand, within a *item-based* type of recommendation [3, 6], the target item i is associated with a score

$$h^S_{ui} = \sum_{j \in \mathcal{I}} f(w_{ij}) r_{uj} = \sum_{j \in \mathcal{I}(u)} f(w_{ij}),$$

and hence, the score is proportional to the similarities between item i and other items already purchased by the user u ($j \in \mathcal{I}(u)$).

[1] University of Padova, Italy, email: aiolli@math.unipd.it

The function $f(w)$ can be assumed monotonic not decreasing and its role is to emphasize/deemphasize similarity contributions in such a way to adjust the *locality* of the scoring function, that is how many of the nearest users/items really matter in the computation. As we will see, a correct setting of this function turned out to be very useful with the challenge data.

Interestingly, in both cases, we can decompose the user and item contributions in a linear way, that is, we can write $h^U_{ui} = \mathbf{w}_u^\top \mathbf{r}_i$, $\mathbf{w}_u \in \mathbb{R}^n$, and $h^S_{ui} = \mathbf{w}_i^\top \mathbf{r}_u$, $\mathbf{w}_i \in \mathbb{R}^m$. In other words, we are defining an embedding for users (in user based recommendation systems) and for items (in item based recommendation systems). In the specific case above, this corresponds to choose the particular vector $\mathbf{r}_i$ as the vector with n entries in $\{0,1\}$, where $\mathbf{r}_u^{(i)} = r_{ui}$. Similarly, for the representation of users in item-based scoring, we choose $\mathbf{r}_u$ as the vector with m entries in $\{0,1\}$, such that $\mathbf{r}_i^{(u)} = r_{ui}$. The main contribution of this paper is to explore how we can set the vectors $\mathbf{w}_i$ and $\mathbf{w}_u$ in a principled way by using the entire user-item preference matrix on-the-fly when a new recommendation has to be done. Alternatively, we can also try to learn the weight vectors from data by noticing that a recommendation task can be seen as a multilabel classification problem where songs represent the labels and users represent the examples. We have performed preliminary experiments in this sense using the preference learning approach described in [1]. The results are promising but the problem in this case is the computational requirements of a *model-based* paradigm like this. For this reason we decided to postpone a further analysis of this setting to future works.

Finally, an alternative and useful way to see at the duality between user-based and item-based recommendation is the following that highlight a clear connection between this task and link prediction. Specifically, we can think of the user-item matrix $R = \{r_{ui}\}$ as a bipartite graph where the set of nodes is $\mathcal{N} = \mathcal{U} \cup \mathcal{I}$, and there exist only edges from $u \in \mathcal{U}$ to $i \in \mathcal{I}$ whenever $r_{ui} = 1$. Now, the user based recommendation strategy corresponds to the intuition that if a user tends to link to the same set of items as the active users, then, this gives us some evidence that can exist a link between the active user and the item i. Dually, in item-based recommendation, the intuition is that, if the active user links to one out of two items which tend to be linked by the same users, then, we can infer that the active user will probably link the other item as well.

2.2 User-based and Song-based similarity

A large part of CF literature in the last decade deals with the problem of defining a good similarity measure. A common opinion is that it cannot exist a single similarity measure that can fit all possible domains where collaborative filtering is used. In this section, we try to define a parametric family of user-based and item-based similarities that can fit different problems.

In the challenge, we have not relevance grades since the ratings are binary values. This is a first simplification we can exploit in the definition of similarity functions. The similarity function that is commonly used in this case, both for the user-based case and the item-based case, is the cosine similarity. In the case of binary grades the cosine similarity can be simplified as in the following. Let $\mathcal{I}(u)$ be the set of items rated by a generic user u, then the cosine similarity between two users u, v is defined by

$$w_{uv} = \frac{|\mathcal{I}(u) \cap \mathcal{I}(v)|}{|\mathcal{I}(u)|^{\frac{1}{2}} |\mathcal{I}(v)|^{\frac{1}{2}}}$$

and, similarly for items, by setting $\mathcal{U}(i)$ the set of users which have rated item i, we have:

$$w_{ij} = \frac{|\mathcal{U}(i) \cap \mathcal{U}(j)|}{|\mathcal{U}(i)|^{\frac{1}{2}} |\mathcal{U}(j)|^{\frac{1}{2}}}.$$

The cosine similarity has the nice property to be symmetric but, as we show in the experimental section, it might not be the better choice. In fact, especially for the item case, we are more interested in computing how likely it is that an item will be appreciated by a user when we *already* know that the same user likes another item. It is clear that this definition is not symmetric. As an alternative to the cosine similarity and, we think, a more well founded way of computing weights w_{ij} in this case, is by resorting to the conditional probability measure which can be estimated with the following formulas:

$$w_{uv} = P(u|v) = \frac{|\mathcal{I}(u) \cap \mathcal{I}(v)|}{|\mathcal{I}(v)|}$$

and

$$w_{ij} = P(i|j) = \frac{|\mathcal{U}(i) \cap \mathcal{U}(j)|}{|\mathcal{U}(j)|}$$

Previous works (see [4] for example) pointed out that the conditional probability measure of similarity, $P(i|j)$, has the limitation that items which are purchased frequently tend to have higher values not because of their co-occurrence frequency but instead because of their popularity. In our opinion, this might not be a limitation in a recommendation setting. Perhaps, this could be an undesired feature when we want to cluster items. In fact, this correlation measure has not to be thought of as a real similarity measure. As we will see, experimental results seem to confirm this hypothesis, at least in the item-based similarity case.

Now, we are able to propose a parametric generalization of the above similarity measures. This parametrization permits us ad-hoc optimizations of the similarity function for the domain of interest. For example, this can be done by validating on available data.

Specifically, we propose to use the following combination of conditional probabilities:

$$w_{uv} = P(v|u)^\alpha P(u|v)^{1-\alpha} \quad w_{ij} = P(j|i)^\alpha P(i|j)^{1-\alpha} \qquad (1)$$

where $\alpha \in [0,1]$ is a parameter to tune. As above, we estimate the probabilities by resorting to the frequencies in the data and derive the following:

$$w_{uv} = \frac{|\mathcal{I}(u) \cap \mathcal{I}(v)|}{|\mathcal{I}(u)|^\alpha |\mathcal{I}(v)|^{1-\alpha}} \quad w_{ij} = \frac{|\mathcal{U}(i) \cap \mathcal{U}(j)|}{|\mathcal{U}(i)|^\alpha |\mathcal{U}(j)|^{1-\alpha}}. \qquad (2)$$

It is easy to note that the standard similarity based on the conditional probability $P(u|v)$ (resp. $P(i|j)$) is obtained setting $\alpha = 0$, the other inverted conditional $P(v|u)$ (resp. $P(j|i)$) is obtained setting $\alpha = 1$, and, finally, the cosine similarity case is obtained when $\alpha = \frac{1}{2}$. This analysis also suggests an interesting interpretation of the cosine similarity on the basis of conditionals.

2.3 Locality of the Scoring Function

In Section 2 we have seen that, in memory based CF, the final recommendation is computed by a scoring function which aggregates the scores obtained using individual users or items. So, it is important to determine how much each individual scoring component influences the overall scoring. This is the role of the function $f(w)$. In the following experiments we use the exponential family of functions, that is $f(w) = w^q$ where $q \in \mathbb{N}$. The effect of this exponentiation is the following. When q is high, smaller weights drop to zero while higher ones are (relatively) emphasized. At the other extreme, when $q = 0$, the aggregation is performed by simply adding up the ratings. We can note that, in the user-based type of scoring function, this corresponds to take the popularity of an item as its score, while, in the case of item-based type of scoring function, this would turn out in a constant for all items (the number of ratings made by the active user).

2.4 Ranking Aggregation

There are many sources of information available regarding songs. For example, it could be useful to consider the additional meta-data which are also available and to construct alternative rankings based on that. It is always difficult to determine a single strategy which is able to correctly rank the songs. An alternative is to use multiple strategies, generate multiple rankings, and finally combine those rankings. Typically, these different strategies are individually *precision oriented*, meaning that each strategy is able to correctly recommend a few of the correct songs with high confidence but, it may be that, other songs which the user likes, cannot be suggested by that particular ranker. Hopefully, if the rankers are different, then the rankers can recommend different songs. If this is the case, a possible solution is to predict a final recommendation that contains all the songs for which the single strategies are more confident. A stochastic version of this aggregation strategy can be described in the following way. We assume we are provided with the list of songs not yet rated by the active user in order of confidence for all the available strategies. On each step, the recommender randomly choose one of the lists according to a probability distribution p_i over the predictors and recommends the best scored item of the list which has not yet been inserted in the current recommendation.

3 Experiments and Results

In the MSD challenge we have: *i)* the full listening history for about 1M users, *ii)* half of the listening history for 110K users (10K validation set, 100K test set), and we have to predict the missing half. Further, we also prepared a "home-made" validation subset (*HV*) of the original training data of about 900K users of training (*HVtr*, with full listening history). The remaining 100K user's histories has been split in two halves (*HVvi* the visible one, *HVhi* the hidden one).

The experiments presented in this section are based on this *HV* data and compare different similarities and different approaches. The baseline is represented by the simple popularity based method which recommends the most popular songs not yet listened to by the user. Besides the baseline, we report experiments on both the user-based and song-based scoring functions, and an example of the application of ranking aggregation.

3.1 Taste Profile Subset Stats

For completeness, in this section, we report some statistics about the original training data. In particular, the following table shows the minimum, maximum, and average, number of users per song and songs per user. The median value is also reported.

.	min	max	ave	median
users per song	1	110479	125.794	13
songs per user	10	4400	47.45681	27

We can see that the large majority of songs have only few users which listened to it (less than 13 users for half of the songs) and the large majority of users have listened to few songs (less than 27 for half of the users). These characteristics of the dataset make the top-τ recommendation task quite challenging.

3.2 Truncated Mean Average Precision

Conformingly to the challenge, we used the truncated mAP (mean average precision) as the evaluation metric. Let y denote a ranking over items, where $y(p) = i$ means that item i is ranked at position p. The mAP metric emphasizes the top recommendations. For any $k \leq \tau$, the *precision at k* (π_k) is defined as the proportion of correct recommendations within the top-k of the predicted ranking (assuming the ranking y does not contain the visible songs),

$$\pi_k(u, y) = \frac{1}{k} \sum_{p=1}^{k} r_{uy(p)}$$

For each user the (truncated) average precision is the average precision at each recall point:

$$AP(u, y) = \frac{1}{\tau_u} \sum_{p=1}^{\tau} \pi_k(u, y) r_{uy(p)}$$

where τ_u is the smaller between τ and the number of user u's positively associated songs.

The average of $AP(u, y_u)$'s over all users gives the mean average precision (mAP).

3.3 Results

The result obtained on the HV data with the baseline (recommendation by popularity) is presented in Table 1. With this strategy, each song i simply gets a score proportional to the number of users $|\mathcal{U}(i)|$ which listened to the song.

Baseline (Recommendation by Popularity)	0.02262

Table 1: mAP@500 obtained by the baseline method (song popularity) on HV data.

In Table 2, we report on experiments that show the effect of the locality parameter q for different strategies: item based and user based (both conditional probability and cosine versions). As we can see, beside the case IS with cosine similarity (Table 2b), a correct setting of the parameter q drammatically improves the effectiveness on HV data. We can clearly see that the best performance is reached with the conditional probability on an item based strategy (Table 2a).

In Figure 1, we present results obtained fixing the parameter q and varying the parameter α for both user-based and item-based recommendation strategies. We see that, in the item-based case, the results improve when setting a non-trivial α. In fact, the best result has been obtained for $\alpha = 0.15$.

Finally, in Table 3, two of the best performing rankers are combined, and their recommendation aggregated, by using the stochastic algorithm described in Section 2.4. In particular, in order to maximize the diversity of the two rankers, we aggregated an item-based ranker with a user-based ranker. We can see that the combined performance improves further on validation data. Building alternative and effective rankers based on available meta-data is not a trivial task and it was not the focus of our current study. For this we decided to postpone this additional analysis to a near future.

IS ($\alpha = 0$)	mAP@500
q=1	0.12224
q=2	0.16581
q=3	**0.17144**
q=4	0.17004
q=5	0.16830

(a)

IS ($\alpha = \frac{1}{2}$)	mAP@500
q=1	**0.16439**
q=2	0.16214
q=3	0.15587
q=4	0.15021
q=5	0.14621

(b)

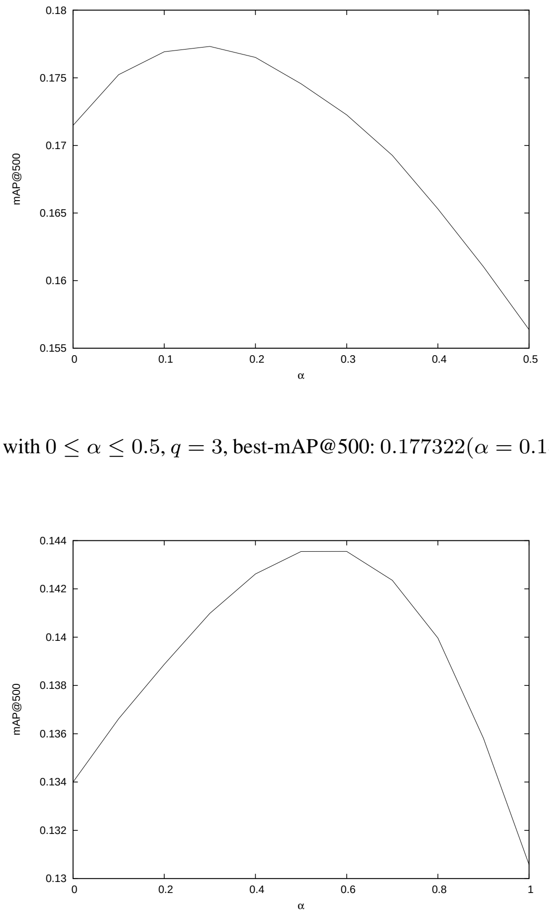
US ($\alpha = 0$)	mAP@500
q=1	0.08030
q=2	0.10747
q=3	0.12479
q=4	0.13298
q=5	**0.13400**
q=6	0.13187
q=7	0.12878

(c)

US ($\alpha = \frac{1}{2}$)	mAP@500
q=1	0.07679
q=2	0.10436
q=3	0.12532
q=4	0.13779
q=5	0.14355
q=6	**0.14487**
q=7	0.14352

(d)

Table 2: Results obtained by item-based (IS) and user-based (US) CF methods varying the locality parameter (exponent q) of the similarity function.



(a) IS with $0 \leq \alpha \leq 0.5$, $q = 3$, best-mAP@500: $0.177322(\alpha = 0.15)$

(b) US with $0 \leq \alpha \leq 1$, $q = 5$, best-mAP@500: $0.143551(\alpha = 0.6)$

Figure 1: Results obtained by item-based (IS) and user-based (US) CF methods varying the α parameter.

(IS, $\alpha = 0.15, q = 3$)	(US, $\alpha = 0.3, q = 5$)	mAP@500
0.0	1.0	0.14098
0.1	0.9	0.14813
0.2	0.8	0.15559
0.3	0.7	0.16248
0.4	0.6	0.16859
0.5	0.5	0.17362
0.6	0.4	0.17684
0.7	0.3	0.17870
0.8	0.2	**0.17896**
0.9	0.1	0.17813
1.0	0.0	0.17732

(a)

Table 3: Results obtained aggregating the rankings of two different strategies, item-based (IS, $\alpha = 0.15$, $q = 3$) and user-based (US, $\alpha = 0.3$, $q = 5$), with different combinations.

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments, which helped improve this paper considerably.

REFERENCES

[1] Fabio Aiolli and Alessandro Sperduti, ‘A preference optimization based unifying framework for supervised learning problems’, in *Preference Learning*, eds., Johannes Fürnkranz and Eyke Hüllermeier, 19–42, Springer-Verlag, (2010).

[2] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere, ‘The million song dataset’, in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, (2011).

[3] Mukund Deshpande and George Karypis, ‘Item-based top-*n* recommendation algorithms’, *ACM Trans. Inf. Syst.*, **22**(1), 143–177, (2004).

[4] George Karypis, ‘Evaluation of item-based top-n recommendation algorithms’, in *CIKM*, pp. 247–254, (2001).

[5] Brian McFee, Thierry Bertin-Mahieux, Daniel P.W. Ellis, and Gert R.G. Lanckriet, ‘The million song dataset challenge’, in *Proceedings of the 21st international conference companion on World Wide Web*, WWW ’12 Companion, pp. 909–916, New York, NY, USA, (2012). ACM.

[6] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl, ‘Item-based collaborative filtering recommendation algorithms’, in *WWW*, pp. 285–295, (2001).

[7] Xiaoyuan Su and Taghi M. Khoshgoftaar, ‘A survey of collaborative filtering techniques’, *Advances in Artificial Intelligence*, (January 2009).

Using and Learning GAI-Decompositions for Representing Ordinal Rankings

Damien Bigot[1], **Hélène Fargier**[1], **Jérôme Mengin**[1], **Bruno Zanuttini** [2 3]

Abstract. We study the use of GAI-decomposable utility functions for representing ordinal rankings on combinatorial sets of objects. Considering only the relative order of objects leaves a lot of freedom for choosing a particular utility function, which allows one to get more compact representations. We focus on the problem of learning such representations, and give a polynomial PAC-learner for the case when a constant bound is known on the degree of the target representation. We also propose linear programming approaches for minimizing such representations.

1 INTRODUCTION

The development of interactive systems for supporting decision-making and of recommender systems highlighted the need for models capable of using a user's preferences to guide her choices. For over fifteen years, the modelling and compact representation of preferences have been active topics in Artificial Intelligence [15, 16, 5, 6, 11].

Existing formalisms are rich and flexible enough to describe the behaviour of complex decision rules. However, for being interesting in practice, these formalisms must also permit fast elicitation of a user's preferences, involving a reasonable amount of interaction only. Configuration of combinatorial products in *business-to-customer* problems [14] and preference-based search [18] are good examples of decision problems in which the user's preferences are not known *a priori*. In such applications, a single interaction with the user must typically last at most 0.25 s, and the whole session must typically last at most 20 minutes, even if the object to be recommended to the user is searched for in a combinatorial set.

When the user's preferences are qualitative and have a "simple" structure (for instance, when they are separable), conditional preference networks (CP-nets) and their variants [5, 4, 6] are popular representation frameworks. In particular, CP-nets come with efficient algorithms for finding most preferred extensions of objects (*outcome optimisation problem*), and with efficient elicitation procedures [13]. Unfortunately, CP-nets suffer a lack of expressivity, since most complete pre-orders cannot be represented by simple (acyclic) CP-nets.

Contrastingly, generalised additively independent decompositions (GAI-decompositions) of utility functions [9, 1, 11] can represent complete pre-orders in a compact way, by exploiting the independencies between sets of variables. In a word, these are representations of utility functions by sums of the form $\sum_{i=1}^{n} u_i(Z_i)$, where the u_i's are sub-utility functions with (hopefully) small scopes Z_i.

Typically, a GAI-decomposition is used to represent a *utility function*, which assigns a value to each possible object and hence, implicitly defines a complete pre-order on them (the greater the value, the more preferred the object). Such values may in some cases represent an amount of money which the user is ready to spend for the object, or may be defined implicitly by preferences on lotteries. However, in many applications, the actual values of the utility function are not important: it is the ranking of the objects that is induced by the utility function, and the properties of this ranking, that are interesting. Furthermore, the representation of the utility function is important too: it should enable fast answers for a variety of queries, not only dominance queries like "Is object o preferred to object o' ?", but also more complex queries like "What are the top-k objects that fulfil some given constraints?", useful for typical recommendation systems.

In this paper, we investigate the use of GAI-decompositions for representing such ordinal rankings. Precisely, we take a GAI-decomposition to represent the ranking defined by the associated utility function. Since in general many different utility functions represent the same ranking, this leaves a lot more freedom for finding compact decompositions than if the utility function is fixed.

In this context, we focus on the problem of learning a compact GAI-decomposition from (ordinal) examples, that is, essentially from statements of the form "I prefer object o to object o'". While some works on this topic have focused on a fixed target *utility function* (rather than a ranking) and assumed the structure (the scopes X_i) to be known in advance, we consider the issue of learning *any* utility function which induces the target *ranking*, and assume nothing about the target structure except for a constant, known bound on its degree. We aim at finding a simple structure, in order to ease optimization queries (among others). This issue has not been addressed in the "learning to rank" literature, where the aim is usually to find a ranking function that can be used to answer dominance queries, as in e.g. [12, 10, 8].

After a review of GAI-decompositions (Section 2), we show in Section 3 that the GAI-decompositions consistent with a set of examples can be defined as the feasible solutions of a linear program. We give an efficient PAC-learner for our problem in Section 4, and extend our approach to the problem of learning *minimal* decompositions in Section 5. Some perspectives and are discusset in Section 6.

2 GAI-DECOMPOSITIONS

In our context, the *preference relation* (or *preferences*) of a user on a set of objects χ is a complete pre-order $\succeq$, that is, a complete and transitive binary relation. Given two objects $o, o' \in \chi$, we take $o \succeq o'$ to mean that o is at least as interesting to the user as o'. We write $\succ$ for the asymmetric part of the relation $\succeq$, and $\sim$ for its symmetric

[1] IRIT, Univ. Toulouse, France; email: prenom.nom@irit.fr

[2] GREYC, Univ. Caen, France; email: prenom.nom@unicaen.fr

[3] Partially funded by the ANR (projet LARDONS, ANR-2010,BLAN-0215)

part. Hence $\succeq$ is a linear order with possible ties on χ, $\succ$ is the strict part of it, and $\sim$ contains the ties.

Generalized additive independence provides a representation for preferences on combinatorial domains. Hence we assume that the objects in χ are described over a set of n variables $X = \{X_1, \dots, X_n\}$. We write D_i for the (finite) domain of X_i, hence the set of all objects is $\chi = D_1 \times \cdots \times D_n$. Though our results can be extended to arbitrary finite domains, for simplicity of exposition we consider *Boolean* domains, and we write $D_i = \{x_i, \bar{x}_i\}$. Slightly abusing notation, we also write objects of χ as sequences of values instead of as vectors. For instance, with $X = \{X_1, X_2, X_3, X_4\}$, the object $(x_1, \bar{x}_2, x_3, x_4) \in \chi$ will be denoted by $x_1\bar{x}_2x_3x_4$. Finally, for any subset of variables $Z \subseteq X$ and object o, $o[Z]$ denotes the projection of o onto the variables in Z, and given a set of objects $O \subseteq \chi$, $O[Z] = \{o[Z] \mid o \in D_1 \times \cdots \times D_n\}$ denotes the set of projections of elements of O onto Z.

It is easy to see that any complete and transitive preference relation $\succeq$ can be represented by a *utility function* $u : \chi \mapsto \mathbb{R}$ satisfying $o \succeq o' \Leftrightarrow u(o) \geq u(o')$ for all $o, o' \in \chi$. Clearly, since the set χ is combinatorial (it contains 2^n objects), it is impractical to directly elicit or explicitly store the relation $\succeq$ or a representation u. However, in some cases, the utility function u satisfies strong independency properties between attributes [7], so that it can be represented by a set of *local* utility functions $\{u_i : D_i \mapsto \mathbb{R} \mid i = 1, \dots, m\}$ each of arity 1, satisfying $u(o) = \sum_{i=1}^{m} u_i(o[\{X_i\}])$ for all objects o. Such representations are clearly very compact, easy to elicit, and allow for efficiently computing optimal objects. Unfortunately, preference relations seldom satisfy this property of *additive independence*.

Example 1. *Consider the set of variables $X = \{X_1, X_2, X_3\}$ with $D_1 = \{\mathrm{beef}(b), \mathrm{fish}(f)\}$, $D_2 = \{\mathrm{redWine}(r), \mathrm{whiteWine}(w)\}$, $D_3 = \{\mathrm{lemon}(l), \mathrm{mustard}(m)\}$: χ contains 8 possible combinations. Consider the following ordering over χ:*

$$brm \succ brl \succ frm \succ frl \sim bwm \succ bwl \succ fwm \succ fwl$$

It can be represented with the additive utility function u defined by the following tables:

u_1 :

b	5
f	2

u_2 :

r	5
w	1

u_3 :

l	2
m	3

Consider now the following ordering:

$$brm \succ bwm \sim fwl \succ brl \succ fwm \sim frl \succ bwl \succ frm$$

To represent this ordering with an additive utility, we should have $u_1(b) > u_1(f)$ since brm is preferred to frm, and $u_1(b) < u_1(f)$ since fwl is preferred to bwl. However, this ordering can be represented using local utilities over several variables: define for any object o, $u(o) = u_{\{X_1,X_2\}}(o[\{X_1, X_2\}]) + u_{\{X_1,X_3\}}(o[\{X_1, X_3\}])$ where $u_{\{X_1,X_2\}}$ and $u_{\{X_1,X_3\}}$ are defined by:

$u_{\{X_1,X_2\}}$:

b, r	5
f, r	1
b, w	2
f, w	4

$u_{\{X_1,X_3\}}$:

b, l	3
b, m	7
f, l	5
f, m	2

Example 1 shows that some variables may depend on one another, and that in this case the utility function must be decomposed onto *sets* of variables rather than onto singletons.

Definition 1 (GAI-decomposition)**.** *Let $X = \{X_1, ..., X_n\}$ be a set of variables, $\chi = D_1 \times \cdots \times D_n$ be a set of objects, and $u : \chi \mapsto \mathbb{R}$ be a utility function on χ. A* GAI-decomposition *of u is a finite set $G = \{u_{Z_1}, \dots, u_{Z_m}\}$ of utility functions on subsets Z_i of X (*i.e., *$u_{Z_i} : \chi[Z_i] \mapsto \mathbb{R}$) such that $u(o) = \Sigma_{i=1}^{m} u_{Z_i}(o[Z_i])$ holds for all $o \in \chi$. The* degree *of G is defined to be $deg(G) = \max_{i=1,\dots,m} |Z_i|$, where $|Z_i|$ denotes the cardinality of Z_i.*

Definition 2. *Let u be a utility function, and let G be a GAI-decomposition of u. Then u (resp. G) is said to* represent *a preference relation $\succeq$ iff $o \succeq o' \Leftrightarrow u(o) \geq u(o')$ for all $o, o' \in \chi$.*
Considering u (resp. G) as given, we also say that it induces *this relation and denote it by $\succeq_u$ (resp. $\succeq_G$).*

The local utility functions $u_{Z_1}, \dots, u_{Z_m}$ are also called *GAI-tables*, because they are typically implemented in tabular form.

Clearly, for any utility function u, $\{u\}$ is a GAI-decomposition of u of degree $|X|$. Also, writing u_c for the constant function with value c, for any GAI-decomposition G of u and any set of variables $Z \subseteq X$, $G \cup \{u_0(Z)\}$ is also a GAI-decomposition of u. Similarly, $G \cup \{u_c(Z)\}$ is a GAI-decomposition of $u + c$, and hence induces the same preferences as G. However, the most interesting decompositions are those which properly refine u.

Definition 3 (utility-preserving refinement)**.** *Let G, G' be two GAI-decompositions of the same utility function u. Then $G = (u_{Z_1}, \dots, u_{Z_m})$ is said to u-*refine *$G' = (u'_{Z'_1}, \dots, u'_{Z'_{m'}})$ if for $i = 1, \dots, m'$, there is $j \in \{1, \dots, m\}$ with $Z'_i \subseteq Z_j$.*

Definition 4 (preference-preserving refinement)**.** *Let G, G' be two GAI-decompositions of utility functions u, u', respectively. Then $G = (u_{Z_1}, \dots, u_{Z_m})$ is said to* refine *$G' = (u'_{Z'_1}, \dots, u'_{Z'_{m'}})$ if $\succeq_G$ and $\succeq_{G'}$ are the same relation and for $i = 1, \dots, m'$, there is $j \in \{1, \dots, m\}$ with $Z'_i \subseteq Z_j$.*

For both definitions, the refinement is said to be *proper* if moreover, for one relation $Z'_i \subseteq Z_j$ as in the definition it holds $Z'_i \neq Z_j$, or for one Z_j there is no $Z'_i \subseteq Z_j$.

Refinement differs from u-refinement because the same preference relation can be represented by several utility functions. We will pay a particular attention to the maximally refined GAI-decompositions which represent a given preference relation.

Example 2. *Consider the set of boolean variables $X = \{X_1, X_2, X_3, X_4\}$. Let u be the GAI utility defined as the sum of $u_{X_1X_2}$, $u_{X_1X_3}$ and $u_{X_1X_4}$, where these sub-utilities are defined by the following tables:*

x_1x_2	9
$x_1\bar{x}_2$	5
$\bar{x}_1x_2$	5
$\bar{x}_1\bar{x}_2$	2

x_1x_3	8
$x_1\bar{x}_3$	9
$\bar{x}_1x_3$	6
$\bar{x}_1\bar{x}_3$	9

x_1x_4	5
$x_1\bar{x}_4$	2
$\bar{x}_1x_4$	4
$\bar{x}_1\bar{x}_4$	1

It can easily be checked that the order over χ induced by u is also induced by the utility u' defined as the sum of $u'_{X_1X_2}$ and $u'_{X_2X_3X_4}$ with the following tables:

x_1x_2	6
$x_1\bar{x}_2$	2
$\bar{x}_1x_2$	1
$\bar{x}_1\bar{x}_2$	0

$x_2x_3x_4$	3	$\bar{x}_2x_3x_4$	2
$x_2x_3\bar{x}_4$	0	$\bar{x}_2x_3\bar{x}_4$	0
$x_2\bar{x}_3x_4$	7	$\bar{x}_2\bar{x}_3x_4$	4
$x_2\bar{x}_3\bar{x}_4$	1	$\bar{x}_2\bar{x}_3\bar{x}_4$	1

Using a small program based on the ideas developed in the next section, we have checked that none of these two GAI-decompositions of the same pre-order can be refined.

This example shows that there is not always a unique maximally refined GAI-decomposition inducing a given pre-order.

3 REPRESENTATION OF EXAMPLES

Our aim in this paper is to learn GAI-decompositions which induce a hidden target preference relation. Hence in the following we assume that there is a set of Boolean variables $X = \{X_1, \ldots, X_n\}$, which defines a set of objects χ, and a target preference relation $\succeq$ on χ, hidden to the learner. The learner has access to information on $\succeq$ through examples.

Definition 5 (example). *An example e of $\succeq$ is a triple of the form (o, R, o'), where $o, o' \in \chi$ and R is one of $\succ, \succeq, \sim, \preceq, \prec$.*
For a set of examples E, we write O_E for the set of all objects involved in at least one example of E.

Examples formalize the information received by the learner, especially by observing the user. For instance, if the learner observes that the user always chooses o over o', it may represent this as the example $(o, \succ, o')$. Similarly, if the user sometimes chooses o over o', and sometimes o' over o, this may be represented as the example $(o, \sim, o')$, etc.

Definition 6 (consistency). *A GAI-decomposition G of a utility function u is said to be* consistent *with a set of examples E if for every example $(o, R, o') \in E$, $u(o) > u(o')$ (respectively $u(o) \geq u(o'), u(o) = u(o'), \ldots$) holds if R is the relation $\succ$ (respectively $\succeq, \sim, \ldots$).*

Clearly, given a constant k, there is not always a GAI-decomposition of degree k (or less) which is consistent with a given set of examples E. To formalize this, we define a set of examples E to be *k-sound* if there is at least one utility function u and a decomposition G of u, of degree at most k, that is consistent with E.

We now define a system of linear inequalities, whose solutions essentially correspond to the GAI-decompositions of degree k consistent with E.

Definition 7 (linear representation of an example). *Let $e = (o, R, o')$ be an example of the target preference relation $\succeq$, and let $k \in \mathbb{N}$. Moreover, let $\sigma > 0$ be a real constant, positive but arbitrary. Finally, for all subsets of variables $Z \subseteq X$ with $0 < |Z| \leq k$ and assignments z to Z, let $U_{Z,z}$ be a formal variable.*

The linear inequality *for $e = (o, R, o'), k, \sigma$, written $ineq_k^\sigma(E)$ (or simply $ineq_k(E)$) is defined to be*

$$\sum_{Z \subseteq X, 0 < |Z| \leq k} U_{Z,o[Z]} \geq \sigma + \sum_{Z \subseteq X, 0 < |Z| \leq k} U_{Z,o'[Z]}$$

if R is the relation $\succ$, to be

$$\sum_{Z \subseteq X, 0 < |Z| \leq k} U_{Z,o[Z]} \geq \sum_{Z \subseteq X, 0 < |Z| \leq k} U_{Z,o'[Z]}$$

if R is the relation $\succeq$, and similarly for the relations $\sim$ (using $=$ in $ineq_k(E)$), $\preceq$ (using $\leq$), and $\prec$ (using $\leq$ and σ).

Definition 8 (linear system). *Let E be a set of examples of the target preference relation $\succeq$, and let $k \in \mathbb{N}$, $\sigma > 0$. The* linear system *for E, k, σ is defined to be the conjunction of linear inequalities $\Sigma_k^\sigma(E) = \bigwedge_{e \in E} ineq_k^\sigma(e)$ (also written simply $\Sigma_k(E)$).*

Intuitively, variables $U_{Z,z}$ encode the components of the GAI-tables in a decomposition G of the target relation. We use a constant σ for strict preference with the aim of using linear programming, for which we need a closed topological space. Proposition 1 below shows that this is without loss of generality.

Importantly, note that the system $\Sigma_k(E)$ has at most $\sum_{i=0}^{k} 2^i \binom{n}{i}$ variables (as many as possible assignments to subsets of at most k variables). However, another bound is obtained by observing that the variable $U_{Z,z}$ appears only if there is an object $o \in O_E$ with $o[Z] = z$. Hence the number of variables occurring in $\Sigma_k(E)$ is at most $\sum_{i=0}^{k} \binom{n}{i} . |O_E|$. Whatever formula we use, provided k is bounded by a constant, the size of $\Sigma_k(E)$ is polynomial in the number of variables n and the number of examples E (using $|O_E| \leq 2|E|$).

Example 3. *Let $o = x_1 x_2 \bar{x}_3$ and $o' = \bar{x}_1 x_2 x_3$. The linear inequality associated with the exemple $e = (o, \succ, o')$ for $k = 2$ and $\sigma = 0.1$ is (writing, for exemple, $U_{x_1 \bar{x}_2}$ for $U_{\{X_1, X_2\}, x_1 \bar{x}_2}$) :*

$$\begin{aligned} & U_{x_1} + U_{x_2} + U_{\bar{x}_3} + U_{x_1 x_2} + U_{x_1 \bar{x}_3} + U_{x_2 \bar{x}_3} \\ \geq \ & U_{\bar{x}_1} + U_{x_2} + U_{x_3} + U_{\bar{x}_1 x_2} + U_{\bar{x}_1 x_3} + U_{x_2 x_3} + 0.1 \end{aligned}$$

We now show that the linear system $\Sigma_k(E)$ characterizes the GAI-decompositions of degree at most k and consistent with E. For technical reasons, we restricted ourselves to utility functions u with *span at least σ*, that is, satisfying $|u(o) - u(o')| \geq \sigma$ for all o, o' with $u(o) \neq u(o')$. This is without loss of generality however, since if u has span $\sigma_u < \sigma$, then u', defined by $u'(o) = \frac{\sigma}{\sigma_u} u(o)$ for all $o \in \chi$, is consistent with E as well and has span σ.

Proposition 1. *Let $\succeq$ be a preference relation on χ, let E be a set of examples for $\succeq$, and let $k \in \mathbb{N}$, $\sigma > 0$. Then the GAI-decompositions of degree at most k, span of at least σ, and consistent with E are exactly the solutions of $\Sigma_k(E)$.*

4 LEARNING

In this section we give an algorithm which, given a constant $k \in \mathbb{N}$ and a k-sound, hidden target preference relation $\succeq$, learns a GAI-decomposition G of $\succeq$ from examples only, in the *Probably Approximately Correct* (PAC) framework [17] (see Section 4.2). Our algorithm essentially maintains the *version space* of all GAI-decompositions of degree k (and span at least σ) consistent with the examples received so far, using a compact representation by $\Sigma_k(E)$.

4.1 VC-Dimension

So as to study the number of examples needed to learn $\succeq$, we first study the Vapnik-Chervonenkis dimension (VC-dimension for short) of the class G_k of all relations $\succeq$ which can be represented by a GAI-decomposition of degree at most k.

The VC-dimension concerns classes of binary concepts, that is, concepts c which assign one of two values to any object x (values $c(x)$ and $\neg c(x)$). Hence we view $\succeq$ as the two binary concepts $\succ$ and $\prec$ over objects $(o, o') \in \chi \times \chi$ (and $G_k^{\succ}, G_k^{\prec}$ denote the corresponding classes of concepts). This gives an equivalent view since, for instance, $o \succeq o'$ is equivalent to $o \not\prec o'$, $o \sim o'$ is equivalent to $o \not\prec o' \wedge o \not\succ o'$, etc. Intuitively, the VC-dimension of $\succ$ is the largest number of "independent" couples (o, R, o'), in the sense that the relation R of none depends on the relation of the others.

Definition 9 (VC-dimension). *Let C be a set of binary concepts over $\chi \times \chi$. A set of couples $O \subseteq \chi \times \chi$ is said to be* shattered *by C if for any partition $\{O^+, O^-\}$ of O, there is a concept $c \in C$ satisfying $\forall (o, o') \in O^+, c(o, o')$ and $\forall (o, o') \in O^-, \neg c(o, o')$. The* VC-dimension *of C is the size of the largest set O that is shattered by C.*

We now give the VC-dimension of classes $G_k^{\succ}, G_k^{\prec}$. The fact that it is polynomial could not be taken for granted even for constant k, since *a priori* arbitrary values can occur in each entry of the GAI-tables.

Proposition 2. *The VC-dimension of $G_k^{\succ}$ (resp. $G_k^{\prec}$) is in $O(2^k n^{k+1})$, where n denotes the number of variables over which the objects are defined.*

Proof Let $K = \sum_{i=0}^{k} 2^i \binom{n}{i}$ (hence $K \in O(2^k n^{k+1})$). We show that no set $O \subseteq \chi \times \chi$ containing more than K couples (o, o') is shattered, from what the claim will follow. By duality, we give the proof for $G_k^{\succ}$.

So let $O \subseteq \chi \times \chi$ with $|O| \geq K + 1$. For all couples $(o, o') \in O$ we define the following formal sum, with variables $U_{Z,z}$ as in Definition 7:

$$V^k_{o,o'} = \sum_{Z \subseteq X, 0 < |Z| \leq k} U_{Z,o[Z]} - U_{Z,o'[Z]}$$

which corresponds to combining the rhs and lhs of any linear inequality associated to o and o'.

All sums $V^k_{o,o'}$ (for $(o, o') \in O$) use variables among the same K variables $U_{Z,z}$ ($0 < |Z| \leq k$). Hence if O contains at least $K + 1$ couples, there is at least one of them, which we write (ω, ω'), such that the sum $V^k_{\omega,\omega'}$ is a linear combination of the others, that is, there are values $\lambda_{o,o'}$ (for $(o, o') \in O \setminus \{(\omega, \omega')\}$) which satisfy

$$V^k_{\omega,\omega'} = \sum_{(o,o') \in O \setminus \{(\omega,\omega')\}} \lambda_{o,o'} V^k_{o,o'}$$

We write $O^{\leq}$ (resp. $O^{>}$) for the set of all couples $(o, o') \in O \setminus \{(\omega, \omega')\}$ with $\lambda_{o,o'} \leq 0$ (resp. $\lambda_{o,o'} > 0$).

First assume $O^{>} \neq \emptyset$. We show that no concept $\succ$ in $G_k^{\succ}$ is consistent with the partition defined by $O^{+} = O^{>}$ and $O^{-} = O^{\leq} \cup \{(\omega, \omega')\}$, that is, no concept $\succ$ satisfies $o \succ o'$ for all $(o, o') \in O^{>}$, $o \not\succ o'$ (*i.e.*, $o \preceq o'$) for all $(o, o') \in O^{\leq}$, and $\omega \not\succ \omega'$. Indeed, given those labels and using Proposition 1, we get that the following linear system must be satisfied (for an arbitrary constant $\sigma > 0$):

$$\begin{aligned} V^k_{o,o'} &\geq \sigma \quad (\forall (o, o') \in O^{>}) \\ V^k_{o,o'} &\leq 0 \quad (\forall (o, o') \in O^{\leq}) \end{aligned}$$

Because of the signs of $\lambda_{o,o'}$'s, it follows that the following inequalities must be satisfied:

$$\begin{aligned} \lambda_{o,o'} V^k_{o,o'} &\geq \lambda_{o,o'} \sigma \quad (\forall (o, o') \in O^{>}) \\ \lambda_{o,o'} V^k_{o,o'} &\geq 0 \quad (\forall (o, o') \in O^{\leq}) \end{aligned}$$

Hence all solutions of this system must satisfy

$$\sum_{(o,o') \in O^{>} \cup O^{\leq}} \lambda_{o,o'} V^k_{o,o'} \geq \sigma \sum_{(o,o') \in O^{>}} \lambda_{o,o'}$$

that is (using $O^{>} \cup O^{\leq} = O \setminus \{(\omega, \omega')\}$),

$$V_{\omega,\omega'} \geq \sigma \sum_{(o,o') \in O^{>}} \lambda_{o,o'}$$

Because of $\sigma > 0$, $O^{>} \neq \emptyset$ and $\lambda_{o,o'} > 0$ for $(o, o') \in O^{>}$, it follows that $\omega \not\succ \omega'$ is impossible, so O is not shattered by $G_k^{\succ}$.

Now assume $O^{>} = \emptyset$. We show that no concept $\succ$ in $G_k^{\succ}$ is consistent with the partition defined by $O^{+} = O^{\leq} \cup \{(\omega, \omega')\}$ and $O^{-} = \emptyset$. Indeed, reasoning as above we get:

$$\begin{aligned} V^k_{o,o'} &\geq 0 \quad (\forall (o, o') \in O^{\leq}) \\ \lambda_{o,o'} V^k_{o,o'} &\leq 0 \quad (\forall (o, o') \in O^{\leq}) \\ \sum_{(o,o') \in O^{\leq}} \lambda_{o,o'} V^k_{o,o'} &\leq 0 \\ V_{\omega,\omega'} \leq 0 & \end{aligned}$$

and hence, $\omega \succ \omega'$ is impossible, showing again that O is not shattered by $G_k^{\succ}$. Since O was arbitrary of size at least K, we conclude that the VC-dimension of $G_k^{\succ}$ is at most K. □

4.2 Algorithm

We now give an algorithm for learning a GAI decomposition of a hidden preference relation $\succeq^*$ accessed through examples. We use the *probably approximately correct learning (PAC learning)* framework proposed by Valiant [17]: the learner asks for a number m of examples (o, R, o') of the target relation $\succeq^*$, and computes a preference relation $\succeq$. The number m of examples in the sample is chosen by the learner as a function of the number of variables n and two real parameters $\varepsilon, \delta \in]0, 1[$. Each example is drawn at random according to a probability distribution D on $\chi \times \chi$; D is fixed but unknown to the learner. For any couple (o, o') drawn from $\chi \times \chi$, the learner receives the example (o, R, o'), where R is determined by $\succeq^*$ (without noise). In this context, an algorithm is a *PAC*-learner if

- it outputs a concept $\succeq$ which with probability at least $1 - \delta$ has error less than ε on couples drawn from $\chi \times \chi$ according to D. Formally, $\sum\{D(o, o') \mid oRo'$ but $\neg(oR^*o')\} < \varepsilon$ holds with probability at least $1 - \delta$, where R is any relation in $\{\succ, \sim, \prec\}$,
- the number m of examples asked by the learner is polynomial in $n, 1/\varepsilon, 1/\delta$,
- the algorithm runs in time polynomial in $n, 1/\varepsilon, 1/\delta$ (counting unit time for asking and receiving an example).

A concept class is said to be *efficiently PAC-learnable* if such a PAC-learner exists for the class.

The framework of PAC-learning captures situations where the learner observes some objects in its environment (those that come to it — it cannot choose which), and is given by a "teacher" the correct labels for these objects. Some objects occurring possibly more seldom than others (as formalized by the distribution D), the learner has less chances to learn with them, but it is less penalized by errors on them.

In order to show that for fixed, constant k, the class G_k of GAI-decompositions having degree at most k are PAC-learnable, we follow the classical *consistent learning* approach. The learner maintains a concept (in fact, the version space of all concepts) consistent with each of the examples received so far, namely, it maintains the linear system $\Sigma_k(E)$ (for a fixed but arbitrary span $\sigma > 0$).

Figure 1 depicts the algorithm. Since $\succeq^*$ is assumed to be k-sound and our setting is noise-free, the algorithm always returns a solution, *i.e.*, $\Sigma_k(E)$ is necessarily a consistent system. The number m of examples which the learner needs is given by the following proposition.

Proposition 3. *For any constant $k > 0$, the class G_k of GAI-decompositions of degree at most k is efficiently PAC-learnable. The number m of examples required by Algorithm* GAI-Learning *is in $O(\max(\frac{1}{\varepsilon} \log \frac{1}{\delta}, \frac{2^k n^{k+1}}{\varepsilon} \log \frac{1}{\varepsilon})$.*

Algorithm 1: The GAI-Learning Algorithm

```
begin
    $\Sigma_k(E) \leftarrow \emptyset$;
    for $i = 1, \dots, m$ do
        ask for an example $e$ of the form $(o, R, o')$;
        add $ineq_k(e)$ to $\Sigma_k(E)$;
    compute a solution $Sol$ of $\Sigma_k(E)$;
    return the GAI-decomposition encoded by $Sol$;
end
```

Proof. Proposition 1 shows that $\Sigma_k(E)$ is solvable, and that the concept returned by the algorithm is consistent with all the examples received.

We determine m using the VC-dimension of $G_k^{\succ}$ and $G_k^{\prec}$. Following [2], because the binary relations $\succ$ and $\prec$ uniquely determine the concept $\succeq$ (see the discussion before Definition 9), we define the "dimension" d of the class of non-binary concepts G_k to be the maximum of the VC-dimension of $G_k^{\succ}$ and $G_k^{\prec}$, hence $d \in O(2^k n^{k+1})$; intuitively, a learner learns $\succ$ and $\prec$ in parallel using the same examples for learning both, and deduces $\succeq$ (for more details we refer the reader to [2]).

Then we can apply the well-known generic result of Blumer et al. [3, Theorem 2.1 (ii)] to get that a number of examples $m \in O(\max(\frac{1}{\varepsilon}\log\frac{1}{\delta}, \frac{d}{\varepsilon}\log\frac{1}{\varepsilon})$ is enough for any concept $\succeq$ consistent with the examples received to be probably approximately correct.

Finally, since m is polynomial in $n, 1/\varepsilon, 1/\delta$ (k is bounded), the size of $\Sigma_k(E)$ is polynomial. Since linear programming is polynomial, the proof is complete. □

5 MINIMIZING GAI-DECOMPOSITIONS

So far we have shown that for any constant k (small in practice), the class G_k of GAI-decompositions with degree at most k is PAC-learnable. However, our solution does not distinguish between a decomposition with degree k and one with degree $k' \ll k$, nor does it distinguish between one with t clusters of variables Z_i and one with $t' \ll t$ clusters, etc.

We now briefly discuss how such parameters can be optimized. The first natural objective is to learn a GAI-decomposition which is maximally u-refined, that is, which cannot be decomposed further while preserving the function.

Lemma 1. *Let u_Z be a utility function with a nontrivial GAI-decomposition $(u_{Z_1}, \dots, u_{Z_m})$. Then $\sum_{z \in D(Z)} u_Z(z) > \sum_i \left(\sum_{z_i \in D(Z_i)} u_{Z_i}(z_i)\right)$ holds.*

Proof. We have by definition of a decomposition

$$\sum_{z \in D(Z)} u_Z(z) = \sum_{z \in D(Z)} \sum_i u_{Z_i}(z[Z_i]) = \sum_i \sum_{z \in D(Z)} u_{Z_i}(z[Z_i])$$

Now because there are $2^{|Z|-|Z_i|}$ assignments to Z which match z on Z_i, it follows:

$$\sum_{z \in D(Z)} u_Z(z) = \sum_i \sum_{z_i \in D(Z_i)} 2^{|Z|-|Z_i|} u_{Z_i}(z_i)$$

Finally, because of $Z_i \subseteq Z$ we have $2^{|Z|-|Z_i|} \geq 1$ for all i, and because the GAI-decomposition is not trivial, we have $2^{|Z|-|Z_i|} > 1$ for at least one i, and hence

$$\sum_{z \in D(Z)} u_Z(z) > \sum_i \sum_{z_i \in D(Z_i)} u_{Z_i}(z_i)$$

as desired. □

Corollary 1. *Let E be a k-sound set of examples. Then minimizing the objective function $\sum_{0<|Z|\leq k} \left(\sum_{z \in D(Z)} u_Z(z)\right)$, under the constraints in $\Sigma_k(E)$ plus the constraint $U_{Z,z} \geq 0$ (for all Z, z), yields a GAI-decomposition $(u_{Z_1}, \dots, u_{Z_m})$ which is consistent with E and in which no u_{Z_i} can be u-refined.*

Proof. Observe that due to the nonnegativity constraint on $U_{Z,z}$'s, the minimum is well-defined. Now towards a contradiction, let $G^* = (u^*_{Z_1}, \dots, u^*_{Z_m})$ be an optimal solution, and let (wlog) $G_1 = (u_{Z_{11}}, \dots, u_{Z_{1p}})$ ($Z_{1i} \subseteq Z_1$) be a nontrivial u-refinement of u_{Z_1}. Define G to be obtained from G^* by replacing $u^*_{Z_1}$ with G_1. Then clearly G is a utility-preserving refinement of G^*, hence both represent the same utility function. It follows that G is also consistent with E and also a feasible solution of the program (in particular, it has the same span $\geq \sigma$). Now by Lemma 1, G has a better value than G^*, which contradicts the optimality of G^*. □

Let $D(Z, E)$ denotes the set of the $z \in D(Z)$ such that there si some $o \in O_E$ with $z = o[Z]$: obviously, for every $z \in D(Z)$, if $z \notin D(Z, E)$ then the minimization will yield $U_{Z,z} = 0$. Therefore, the linear program of Corollary 1 can be expressed as follows:

$$\text{(P1)} \begin{cases} \text{minimize} \displaystyle\sum_{Z \subseteq X, 0<|Z|\leq k, z \in D(Z,E)} U_{Z,z} \\ \text{under constraints} \\ \bullet\ ineq_k(e) \text{ for every } e \in E \\ \bullet\ U_{Z,o[Z]} \geq 0 \text{ for every } Z, o \end{cases}$$

However, though (P1) achieves some kind of minimality, it does not tend to minimize the number of nonempty entries (nonnull values) in the tables, as the following example shows.

Example 4. *Consider two boolean variables X_1, X_2, and let $E = \{x_1x_2 \succ x_1\bar{x}_2\ ,\ x_1\bar{x}_2 \succ \bar{x}_1\bar{x}_2\ ,\ \bar{x}_1\bar{x}_2 \succ \bar{x}_1x_2\}$. Fix $k = 2$, $\sigma = 1$, and consider the following decompositions (u_1, u_{12}) and (u'_{12}), which are both consistent with E:*

u_1 :

x_1	1
$\bar{x}_1$	0

u_{12} :

x_1x_2	2
$x_1\bar{x}_2$	1
$\bar{x}_1\bar{x}_2$	1
$\bar{x}_1x_2$	0

u'_{12} :

x_1x_2	3
$x_1\bar{x}_2$	2
$\bar{x}_1\bar{x}_2$	1
$\bar{x}_1x_2$	0

The decomposition (u'_{12}) has fewer non-zero entries than (u_1, u_{12}), however it can be seen that the latter has a better value for (P1) than the former.

Despite this, we can apply some efficient post-processing to the solution computed for (P1), by reporting the values in the table of Z_i to the table of $Z_j \supset Z_i$, if any.

Clearly, minimizing the number of nonempty entries allows for more efficient storage of the decomposition learnt. In order to minimize this number over all possible GAI-decompositions consistent with the example, one has to resort to mixed-integer programming, using an additional set of 0/1 variables of the form $V_{Z,z}$ (recording whether the entry $u_Z(z)$ is nonempty). This yields the following program (using standard constructs):

$$\text{(P2)}\left\{\begin{array}{l} \text{minimize} \displaystyle\sum_{Z\subseteq X, 0<|Z|\leq k, z\in D(Z,E)} V_{Z,z} \\ \text{under constraints} \\ \bullet\ \mathit{ineq}_k(e) \text{ for every } e \in E \\ \bullet\ U_{Z,o[Z]} \geq 0 \text{ for every } Z, o \\ \bullet\ V_{Z,o[Z]} \geq U_{Z,o[Z]} \text{ for every } Z \subseteq X, 0 < |Z| \leq k, o \in O_E \\ \bullet\ V_{Z,o[Z]} \in \{0,1\} \text{ for every } Z \subseteq X, 0 < |Z| \leq k, o \in O_E \end{array}\right.$$

Note that the constant σ must be small enough so that constraining the $U_{Z,o[Z]}$s to be in the $[0,1]$ interval does not artificially eliminate some solutions.

Finally, a natural objective is to minimize the degree of the GAI-decomposition learnt (given that it will be at most k anyway).

Example 5. *Consider three boolean variables X_1, X_2, X_3, let $k = 3$, $\sigma = 1$, and $E = \{x_1x_2x_3 \succ \bar{x}_1x_2x_3\ ,\ \bar{x}_1x_2\bar{x}_3 \succ \bar{x}_1\bar{x}_2\bar{x}_3\}$. Consider the decompositions (u_{123}) and (u'_1, u'_2) defined as follows:*

$$u_{123} : \begin{array}{|c|c|}\hline x_1x_2x_3 & 1 \\ \bar{x}_1x_2\bar{x}_3 & 1 \\ \mathit{else} & 0 \\ \hline\end{array} \quad\Bigg|\quad u'_1 : \begin{array}{|c|c|}\hline x_1 & 1 \\ \bar{x}_1 & 0 \\ \hline\end{array} \quad u'_2 : \begin{array}{|c|c|}\hline x_2 & 1 \\ \bar{x}_2 & 0 \\ \hline\end{array}$$

Both decompositions are consistent with E. Moreover, (u_{123}) is clearly an optimum of (P1) and of (P2), but it does not have minimal degree, since (u'_1, u'_2) has degree 1.

Again, one could resort to mixed-integer programming to minimize the degree over all decompositions consistent with the examples. Nevertheless, it is clearly a more efficient approach to proceed by exhaustive search (or by dichotomy): if there is a decomposition of degree k, then look for one with degree $k - 1$, etc.

6 CONCLUSION

In this paper, we have shown that any complete preorder on a combinatorial domain, which can be represented by a GAI-decomposition of degree k, can also be seen as the solution of a system of linear equations. For a given k, a set of examples of such a hidden preorder (encoding a preference relation) leads to a linear system whose variables encode the components of the GAI-tables. A judicious choice of an objective function allows to get a minimal decomposition (with various notions of minimality).

With this in hand, we designed an algorithm which learns a GAI-decomposition of a hidden preference relation, provided a constant bound on the degree of such a decomposition is known *a priori*, in the framework of PAC-learning. To our knowledge, this is the first algorithm able to learn GAI-decompositions without being given the structure of the tables (the sets of variables Z_i). Even if we require a constant bound on the degree to be given, the result could not be taken for granted, since the presence of arbitrary values in the GAI-tables makes *a priori* GAI-decompositions of degree k a very expressive class (hence difficult to learn). On the practical side, requiring a small, constant bound typically fits in the applicative context, where (human) users usually have very local preferences.

When the training set is noisy or the chosen bound k is too low, the linear system has no solution. It is simple to relax the system by introducing a *slack variable* δ_e in the left part of each inequality $\mathit{ineq}_k(e)$. Then the sum of the δ_i's is obviously to be minimized (the variant of the algorithm given in the paper corresponds to null δ_i's).

The next development of this work is the design of a good strategy for the choice of the degree of the GAI to be learnt. A simple approach is to follow an increasing strategy, first assuming that variables are independent, then setting $k = 2$ and so on, until a good coverage of the examples is reached. A finer strategy would be to use a tolerance parameter and to analyze the (imperfect) graph learnt for $k = 2$ in order to have a better idea of the dependencies: the knowledge of a clique on a set Y of variables leading to the necessity of the local utility function u_Y. More generally, such further development would imply interleaving two procedures, one being devoted to learning k (or the structure of the GAI), and the other to learning the entries in the tables using linear programming.

Last, but not least, we shall go back to the development of (active) elicitation methods close to the ones used in CP-net learning. The idea is to ask the user a series of question, whose answer allows the learner to infer (in)dependencies between variables [13]. In this context, the equations and variables of the linear system would show up only when necessary, leading to a much more efficient procedure in terms of memory.

Acknowledgments We thank all anonymous reviewers of ECAI 2012 for helpful comments.

REFERENCES

[1] Fahiem Bacchus and Adam Grove, 'Graphical models for preference and utility', in *Proceedings of UAI'95*, pp. 3–10, (1995).

[2] Shai Ben-David, Nicolò Cesa-Bianchi, David Haussler, and Philip M. Long, 'Characterizations of learnability for classes of {0,. . . ,n}-valued functions', *J. Of Computer and System Sciences*, **50**, 74–86, (1995).

[3] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth, 'Learnability and the vapnik-chervonenkis dimension', *J. ACM*, **36**(4), 929–965, (1989).

[4] Craig Boutilier, Fahiem Bacchus, and Ronen I. Brafman, 'Ucp-networks: A directed graphical representation of conditional utilities', in *Proceedings of UAI'01*, (2001).

[5] Craig Boutilier, Ronen Brafman, Holger Hoos, and David Poole, 'Reasoning with conditional ceteris paribus preference statem', in *Proceedings of UAI-99*, pp. 71–80, San Francisco, CA, (1999).

[6] Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole, 'Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements', *JAIR*, **21**, 135–191, (2004).

[7] G. Debreu, 'Topological methods in cardinal utility theory', in *Mathematical Methods in the Social Sciences*, eds., K.J. Arrow, S. Karlin, and P. Suppes, 16–26, Stanford University Press, Stanford, (1960).

[8] Carmel Domshlak and Thorsten Joachims, 'Unstructuring user preferences: Efficient non-parametric utility revelation', in *Proc. of UAI '05*, pp. 169–177, (2005).

[9] P.C. Fishburn, *Utility Theory for Decision Making*, Wiley, New York, 1970.

[10] Yoav Freund, Raj D. Iyer, Robert E. Schapire, and Yoram Singer, 'An efficient boosting algorithm for combining preferences', *Journal of Machine Learning Research*, **4**, 933–969, (2003).

[11] Christophe Gonzales and Patrice Perny, 'Gai networks for utility elicitation', in *Proceedings of KR'04*, pp. 224–234, (2004).

[12] Thorsten Joachims, 'Optimizing search engines using clickthrough data', in *Proc. of ACM KDD'02*, pp. 133–142.

[13] Frédéric Koriche and Bruno Zanuttini, 'Learning conditional preference networks with queries', in *Proc. IJCAI'09*, pp. 1930–1935, (2009).

[14] Daniel Mailharro, 'A classification and constraint-based framework for configuration', *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 383–397, (September 1998).

[15] Anand S. Rao and Michael P. Georgeff, 'Modeling rational agents within a bdi-architecture', in *Proc. KR'92*, pp. 473–484, (1991).

[16] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie, 'Valued constraint satisfaction problems: Hard and easy problems ', in *Proceedings of IJCAI'95)*, pp. 631–637. Morgan Kaufmann, (aot 1995).

[17] Leslie G. Valiant, 'A theory of the learnable', *Communications of the ACM*, **27**(11), 1134–1142, (1984).

[18] Paolo Viappiani, Boi Faltings, and Pearl Pu, 'Preference-based search using example-critiquing with suggestions', *JAIR*, **27**, 465–503, (2006).

Learning Conditional Lexicographic Preference Trees

Michael Bräuning and **Eyke Hüllermeier**[1]

Abstract. We introduce a generalization of lexicographic orders and argue that this generalization constitutes an interesting model class for preference learning in general and ranking in particular. We propose a learning algorithm for inducing a so-called conditional lexicographic preference tree from a given set of training data in the form of pairwise comparisons between objects. Experimentally, we validate our algorithm in the setting of multipartite ranking.

1 INTRODUCTION

Preference learning is an emerging subfield of machine learning that has received increasing attention in recent years [13]. A specific though important special case of preference learning is "learning to rank", that is, the learning of models that can be used to predict preferences in the form of rankings of a set of alternatives [7, 8]. Ranking problems are often reduced to problems of a simpler type, such as learning a value function that assigns scores to alternatives (with better alternatives having higher scores) or learning a binary predicate that compares pairs of alternatives [15]; while the former approach is close to regression, the latter is in the realm of classification learning.

Another approach to learning ranking functions is to proceed from specific model assumptions, that is, assumptions about the structure of the sought preference relations. This approach is less generic than the previous ones, as it strongly depends on the concrete assumptions made. On the other hand, it typically offers the advantage of being more easily understandable and interpretable. An example is the representation of preferences in the form of a CP-net [5]. Another example is lexicographic orders that are widely accepted as a plausible representation of (human) preferences [16], especially in complex decision making domains [1]. Here, the assumption is that the target ranking of a set of alternatives, each one described in terms of multiple attributes, can be represented as a lexicographic order.

From a machine learning point of view, assumptions of the above type can be seen as an inductive bias restricting the hypothesis space. Provided the bias is correct, this is clearly an advantage, as it may simplify the learning problem. On the other hand, an overly strong bias may prevent the learner from approximating the target ranking sufficiently well. For example, while being plausible in some situations, the assumption of a lexicographic order will be too restrictive for many applications.

In this paper, we therefore present a method for learning generalized lexicographic orders. While still being simple and easy to understand, the model class we consider relaxes some of the assumptions of a proper lexicographic order. More specifically, we increase flexibility thanks to two extensions of conventional lexicographic orders.

- First, we allow for *conditioning* [3, 4]: The importance of attributes as well as the preferences for the values of an attribute may depend on the values of other variables preceding that one in the underlying variable order.
- Second, we allow for *grouping* [17]: Several (one-dimensional) variables can be grouped into a single high-dimensional variable, and preferences can be specified on the cartesian product of the corresponding domains.

The remainder of this paper is organized as follows. In the next section, we give a brief overview of related work. In Section 3, we introduce generalized lexicographic orders and the notion of conditional lexicographic preference trees. In Section 4, we present an algorithm for learning such preference models from data. An experimental study is presented in Section 5, prior to concluding the paper in Section 6.

2 RELATED WORK

The use of lexicographic orders in preference modeling has already been considered in the seventies of the last century [10], whereas in machine learning, this type of structure has attracted attention only recently. Flach and Matsubara developed a lexicographic ranker called LexRank, using a linear preference ordering on attributes derived by the odds ratio [12, 11]. Experimentally, they show that LexRank is competitive to decision trees and naive Bayes in terms of ranking performance (AUC).

Further work on learning lexicographic orders was done by Schmitt and Martignon [16], Dombi et al. [9], and Yaman et al. [18]. However, these works are based on rather simplistic assumptions. More general models were studied by Booth et al. [3, 4], and in fact, important parts of our approach (such as conditional importance of attributes and conditional preferences on attribute values) is inspired by these models. Their work remains rather theoretical, however, without a practical realization in terms of an implementation of algorithms or an experimental study with real data.

3 GENERALIZED LEXICOGRAPHIC ORDERS

Formally, we proceed from an attribute-value representation of decision alternatives or objects, i.e., an object is represented as a vector

$$\boldsymbol{o} \in \mathcal{O} = \mathcal{D}(V) = \mathcal{D}(A_1) \times ... \times \mathcal{D}(A_n),$$

where $V = \{A_1, ..., A_n\}$ is the set of attributes (variables) and $\mathcal{D}(A_i)$ is the domain of attribute A_i. For a subset $A = \{A_{i_1}, \ldots, A_{i_k}\} \subset V$ of attributes we define $\mathcal{D}(A) = \mathcal{D}(A_{i_1}) \times ... \times \mathcal{D}(A_{i_k})$.

An *assignment* or *instantiation* of a subset $A \subseteq V$ of attributes is an element $\boldsymbol{a} \in \mathcal{D}(A)$; an assignment is called *complete* if $A =$

[1] Department of Mathematics and Computer Science, University of Marburg, Germany, email: {braeunim,eyke}@mathematik.uni-marburg.de

V, otherwise it is called *partial*. For an object $\boldsymbol{o} \in \mathcal{O}$ and a subset $A \subset V$, we denote by $\boldsymbol{o}[A]$ the projection of $\boldsymbol{o}$ from $\mathcal{D}(V)$ to $\mathcal{D}(A)$; if $A = \{A_k\}$ is a single attribute, we also write $\boldsymbol{o}[k]$ instead of $\boldsymbol{o}[\{A_k\}]$.

A lexicographic order on $\mathcal{O}$ is a total order $\succ$ defined in terms of

- a total order $\sqsupset$ on V, i.e., a ranking of the attributes,
- a total order $\sqsupset_i$ on each attribute domain $\mathcal{D}(A_i)$.

More specifically, $\boldsymbol{o}^* \succ \boldsymbol{o}$ (suggesting that $\boldsymbol{o}^*$ is preferred to $\boldsymbol{o}$) if and only if there exists a $k \in \{1, \ldots, n\}$ such that

$$\left(\boldsymbol{o}^*[k] \sqsupset_k \boldsymbol{o}[k]\right) \wedge \Big(\left(A_i \sqsupset A_k\right) \Rightarrow \left(\boldsymbol{o}^*[i] = \boldsymbol{o}[i]\right)\Big)$$

for all $i \in \{1, \ldots, n\}$. The relations $\sqsupset_i$ indicate preference on individual attributes: $a \sqsupset_i b$ means that, for $a, b \in \mathcal{D}(A_k)$, a is preferred to b as a value for attribute A_i. Moreover, the relation $\sqsupset$ reflects the importance of attributes: $A_i \sqsupset A_j$ means that attribute A_i is more important than A_j, whence the former is considered prior to the latter. Without loss of generality, we shall subsequently assume that $A_1 \sqsupset A_2 \sqsupset \cdots \sqsupset A_n$ (unless otherwise stated).

3.1 Conditional preferences on attribute values

Conventional lexicographic orders assume that preferences $\sqsupset_k$ on attribute domains are independent of each other. Needless to say, this assumption is often violated in practice. For example, although it is possible that a person prefers red wine to white wine *in general*, it is also plausible that her preference for wine may depend on the main dish: red is preferred to white in the case of meat, whereas white is preferred to red in the case of fish.

In order to capture attribute dependencies of that type, the preferences relations $\sqsupset_k$ can be conditioned on the values of the attributes A_j preceding A_k in the order $\sqsupset$ [3, 4]. That is, $\sqsupset_k$ is now replaced by a set of strict orders

$$\Big\{ \sqsupset_k^{(a_1,\ldots,a_{k-1})} \mid (a_1, \ldots, a_{k-1}) \in \mathcal{D}(\{A_1, \ldots, A_{k-1}\}) \Big\}$$

Moreover, the order relation $\succ$ on $\mathcal{O}$ is then defined as follows: $\boldsymbol{o}^* \succ \boldsymbol{o}$ for $\boldsymbol{o}^* = (a_1^*, \ldots, a_n^*)$ and $\boldsymbol{o} = (a_1, \ldots, a_n)$ if and only if there exists a $k \in \{1, \ldots, n\}$ such that

$$\Big(\forall\, i \in \{1, \ldots, k-1\} : a_i^* = a_i\Big) \wedge \left(a_k^* \sqsupset_k^{(a_1,\ldots,a_{k-1})} a_k\right).$$

3.2 Conditional attribute importance

Going one step further, one may assume that the values of the first attributes in the attribute order $\sqsupset$ do not only influence the preferences on the values of the attributes that follow, but also the importance of the attributes themselves [3, 4]. Thus, we are no longer dealing with a lexicographic order in the sense that $\sqsupset$ defines a *sequence* of the attributes V according to their importance. Instead, we are dealing with a *tree-like* structure. This structure is defined by the following (choice) function:

$$A = C\Big((A_{i_1}, A_{i_2}, \ldots, A_{i_k}), (a_{i_1}, a_{i_2}, \ldots, a_{i_k})\Big),$$

where $(A_{i_1}, A_{i_2}, \ldots, A_{i_k}) \in V^k$ is a sequence of attributes (such that $A_{i_j} \neq A_{i_k}$ for $j \neq k$) and $a_{i_j} \in \mathcal{D}(A_{i_j})$ for all $j \in \{1, \ldots, k\}$. Moreover, $A \in V \setminus \{A_{i_1}, \ldots, A_{i_k}\}$ is the most important attribute given that $A_{i_j} = a_{i_j}$ for all $j \in \{1, \ldots, k\}$.

3.3 Variable grouping

Another extension consists of grouping several variables, that is, to allow the expression of preferences on *attribute tuples* instead of single attributes only [17]. Formally, this means selecting an index set $\mathcal{I} \subseteq \{1, \ldots, n\}$ and defining a total order relation $\sqsupset_{\mathcal{I}}$ on the cartesian product $\mathcal{D}(V_{\mathcal{I}})$ of the domains $\mathcal{D}(A_i)$, $i \in \mathcal{I}$.

Note that the possibility of variable grouping significantly increases the expressivity of the model class. In particular, taking $\mathcal{I} = \{1, \ldots, n\}$, it is possible to define every order on $\mathcal{D}(V)$, that is, to sort the set of alternatives in any way. Since this level of expressivity is normally not desirable, it is reasonable to restrict to variable grouping of order g_{max}, meaning to impose the constraint $|\mathcal{I}| \leq g_{max}$ for a fixed $g_{max} \leq n$.

3.4 Conditional lexicographic preference trees

Combining the generalizations discussed above, we end up with what we call a Conditional Lexicographic Preference Tree (CLPT). Graphically, this is a tree structure in which

- every node is labeled with a subset of attributes $V_{\mathcal{I}}$ and a total order on the cartesian product $\mathcal{D}(V_{\mathcal{I}})$ of the corresponding attribute domains $\mathcal{D}(A_i)$, $i \in \mathcal{I}$;
- there is one outgoing edge (descendant node) for each value $\boldsymbol{o}[V_{\mathcal{I}}] \in \mathcal{D}(V_{\mathcal{I}})$;
- every attribute $A_i \in V$ occurs at most once on each branch from the root of the tree to a leaf node (i.e., the index sets $\mathcal{I}$ along a branch are disjoint).

We call a CLPT *complete* if every attribute $A_i \in V$ occurs exactly once on each branch from the root of the tree to a leaf node (i.e., the index sets $\mathcal{I}$ along a branch form a partition of $\{1, \ldots, n\}$).

A (complete) CLPT can be thought of a defining an order relation on $\mathcal{O}$ through recursive refinement of a weak order $\succeq$, that is, by refining an order relation with tie groups in a recursive manner (in the following, $\sim$ and $\succ$ denote, respectively, the symmetric and asymmetric part of $\succeq$):

- One starts with a single equivalence class (tie group), i.e., $\boldsymbol{o}^* \sim \boldsymbol{o}$ for all $\boldsymbol{o}^*, \boldsymbol{o} \in \mathcal{O}$.
- Let the root of the CLPT be labeled with the attribute set $V_{\mathcal{I}}$, and let $\sqsupset_{\mathcal{I}}$ denote the corresponding order on $\mathcal{D}(V_{\mathcal{I}})$. The current order $\succeq$ is then refined by letting $\boldsymbol{o}^* \succ \boldsymbol{o}$ whenever $\boldsymbol{o}^*[V_{\mathcal{I}}] \sqsupset_{\mathcal{I}} \boldsymbol{o}[V_{\mathcal{I}}]$; otherwise, if $\boldsymbol{o}^*[V_{\mathcal{I}}] = \boldsymbol{o}[V_{\mathcal{I}}]$, then $\boldsymbol{o}^*$ and $\boldsymbol{o}$ remain tied.
- Thus, a linear order of tie groups (equivalence classes) is produced.
- Each equivalence class (represented by a value $\boldsymbol{a} \in \mathcal{D}(V_{\mathcal{I}})$) is then recursively refined by the subtree the objects of this equivalence class are passed to.

Note that, if the CLPT is complete, the order relation $\succeq$ eventually produced is a total order $\succ$.

4 LEARNING CLPTs

In this section, we outline a method for inducing a CLPT from training data

$$\mathcal{T} = \big\{(\boldsymbol{o}_i^*, \boldsymbol{o}_i)\big\}_{i=1}^N \tag{1}$$

that consists of a set of object pairs $(\boldsymbol{o}_i^*, \boldsymbol{o}_i) \in \mathcal{O}^2$, suggesting that $\boldsymbol{o}_i^*$ is preferred to $\boldsymbol{o}_i$. Roughly speaking, this means finding a CLPT whose induced order relation $\succeq$ on $\mathcal{O}$ is as much as possible in agreement with the pairwise preferences in $\mathcal{T}$ (without overfitting the training data). The induced order relation $\succeq$ is a total order $\succ$ if the CLPT is complete.

4.1 Performance and evaluation measures

In order to evaluate the predictive performance of a CLPT, there is a need to compare the order relation $\succeq$ (with asymmetric part $\succ$) induced by this model with a ground truth order $\succ *$. As will be seen below, the same measures can be used to fit a CLPT to a given set of training data (1) during the training phase. In this case, the "ground truth" is not a total order but a set of pairwise comparisons between objects. Since a total order $\succ^*$ can be decomposed into (a quadratic number of) such comparisons, too, we can assume (without loss of generality) that we compare $\succeq$ with a set $\mathcal{T}$ of pairs $(\boldsymbol{o}^*, \boldsymbol{o}) \in \mathcal{O}^2$, suggesting that $\boldsymbol{o}^*$ should be ranked higher than $\boldsymbol{o}$.

Inspired by the corresponding notions introduced in [6], we define two performance measures of *correctness* and *completeness*, respectively, as follows:

$$\mathrm{CR}(\succeq, \mathcal{T}) = \frac{|C| - |D|}{|C| + |D|}, \quad (2)$$

$$\mathrm{CP}(\succeq, \mathcal{T}) = \frac{|C| + |D|}{|\mathcal{T}|}, \quad (3)$$

where

$$C = \Big\{(\boldsymbol{o}^*, \boldsymbol{o}) \in \mathcal{T} \,|\, \boldsymbol{o}^* \succ \boldsymbol{o}\Big\},$$
$$D = \Big\{(\boldsymbol{o}^*, \boldsymbol{o}) \in \mathcal{T} \,|\, \boldsymbol{o} \succ \boldsymbol{o}^*\Big\}.$$

Note that $\mathrm{CR}(\succeq, \mathcal{T})$ assumes values between -1 (complete disagreement) and $+1$ (complete agreement), while $\mathrm{CP}(\succeq, \mathcal{T})$ ranges between 0 (no comparisons) and 1 (full comparison).

4.2 A greedy learning procedure

We implement an algorithm for learning a CLPT as a (greedy) search in the space of tree structures based on the greedy algorithms presented by Schmitt and Martignon [16] as well as Booth et al. [3, 4]. This is done by constructing the tree from the root to the leaves in a recursive manner. In each step of the recursion, a new node is created with an associated subset $V_\mathcal{I}$ of attributes, where $|V_\mathcal{I}| \leq g_{max}$, and a total order $\sqsupset_\mathcal{I}$ on $\mathcal{D}(V_\mathcal{I})$.

4.2.1 Creating a node

The problem to be solved in each recursion is the following: Given a set of pairwise comparisons $\mathcal{T}$ and a set $V' \subseteq V$ of attributes still available, select a most suitable subset $V_\mathcal{I} \subseteq V'$ and an order $\sqsupset_\mathcal{I}$. Following a greedy strategy, we choose $(V_\mathcal{I}, \sqsupset_\mathcal{I})$ so as to maximize correctness (2), using completeness (3) as a second criterion to break ties.

The selection of an attribute subset $V_\mathcal{I}$ can be done through exhaustive search if its size is sufficiently limited, i.e., if the upper bound g_{max} is small. Otherwise, a complete enumeration of all possibilities may become too expensive. Moreover, for each candidate subset $V_\mathcal{I}$, a total order $\sqsupset_\mathcal{I}$ needs to be determined. Again, all such orders can be tried if $\mathcal{D}(V_\mathcal{I})$ is not too large. Otherwise, heuristic ranking procedures such as Borda count can be used (counting the number of "wins" and "losses" of each value $\boldsymbol{a} \in \mathcal{D}(V_\mathcal{I})$ in the training data $\mathcal{T}$ and sorting according to the difference).

4.2.2 Limiting the number of candidate subsets

In order to avoid a complete enumeration of all candidate subsets $V_\mathcal{I}$ of size $\leq g_{max}$, we combine a greedy search with a kind of lookahead procedure: We provisionally create a node by selecting a single attribute instead of a subset, i.e., we tentatively set g_{max} to 1; apart from that, exactly the same selection procedure (as outlined above) is applied. This step is repeated g_{max} times, thereby producing a subtree of depth g_{max}. Let $V^* \subseteq V$ denote the subset of attributes that occur in this subtree, i.e., that are chosen in at least one of the nodes. Then, as candidate subsets $V_\mathcal{I}$, we only try subsets V^*, i.e., subsets $V_\mathcal{I} \subseteq V^*$ such that $|V_\mathcal{I}| \leq g_{max}$. Obviously, the underlying assumption is that an attribute that has not been chosen in any of the g_{max} steps is not important at this point.

4.2.3 Recursion

Once an optimal subset $V_\mathcal{I}$ has been chosen, the training examples $(\boldsymbol{o}^*, \boldsymbol{o})$ with $\boldsymbol{o}^*[V_\mathcal{I}] \neq \boldsymbol{o}[V_\mathcal{I}]$ are removed from $\mathcal{T}$ (since they are sorted at this node). Moreover, for each value $\boldsymbol{a} \in \mathcal{D}(V_\mathcal{I})$, a data set

$$\mathcal{T}_{\boldsymbol{a}} = \Big\{(\boldsymbol{o}^*, \boldsymbol{o}) \in \mathcal{T} \,|\, \boldsymbol{o}^*[V_\mathcal{I}] = \boldsymbol{o}[V_\mathcal{I}] = \boldsymbol{a}\Big\}$$

is created and passed to the corresponding successor node (together with $V' \setminus V_\mathcal{I}$ as the attributes that have not been used so far). The same recursive procedure is then applied to each of these successor nodes.

4.2.4 Initialization and termination

The learning procedure is called with the original training set $\mathcal{T}$ and the full set V of attributes as candidates. The recursion terminates if no attribute is left ($V' = \emptyset$) or if the set of training examples is empty ($\mathcal{T} = \emptyset$).

4.2.5 CLeRa

We call the algorithm outlined above *CLeRa*, which is short for Conditional Lexicographic Ranker. The CLPT induced by CLeRa can be used to compare new object pairs $\{\boldsymbol{o}^*, \boldsymbol{o}\} \subset \mathcal{O}$. To this end, the tuple is submitted to the root and propagated through the tree until either a leaf node is reached or a node at which $\boldsymbol{o}^*[V_\mathcal{I}] \neq \boldsymbol{o}[V_\mathcal{I}]$; in this case, $\boldsymbol{o}^* \succ \boldsymbol{o}$ is decided if $\boldsymbol{o}^* \sqsupset_\mathcal{I} \boldsymbol{o}$ and $\boldsymbol{o} \succ \boldsymbol{o}^*$ if $\boldsymbol{o} \sqsupset_\mathcal{I} \boldsymbol{o}^*$. Otherwise, if $\boldsymbol{o}^*[V_\mathcal{I}] = \boldsymbol{o}[V_\mathcal{I}]$ in all nodes traversed by the two objects, then $\boldsymbol{o}^* \sim \boldsymbol{o}$.

Given not only a pair but a complete set of objects to be ranked, the pairwise comparison realized by the CLPT can be embedded in any standard sorting algorithm, such as insertion sort. Note that, since $\boldsymbol{o}^* \sim \boldsymbol{o}$ is possible in a pairwise comparison, the result of the sorting procedure will in general only be a weak order $\succeq$.

5 EXPERIMENTAL RESULTS

We evaluate our approach on 15 benchmark data sets from the Statlog and the UCI repository [2]. These data sets, which define binary or ordinal classification problems, were pre-processed as follows: numerical attributes and attributes with more than five values were discretized into four values using equal frequency binning. Moreover, instances with missing values were neglected.

The learning problem we consider is multipartite ranking [14]: Given a set of test instances $X \subset \mathcal{O}$, the goal is to predict a ranking

$\succeq$ that agrees with the (ordered) class labels of these instances. Formally, this agreement is measured in terms of the so-called C-index, which can be seen as an extension of the AUC:

$$C = \frac{1}{\sum_{i<j} n_i n_j} \sum_{1 \le i < j \le m} \sum_{(\boldsymbol{o}, \boldsymbol{o}^*) \in X_i \times X_j} \mathbb{I}(\boldsymbol{o}^* \succ \boldsymbol{o}) + \frac{1}{2}\mathbb{I}(\boldsymbol{o}^* \sim \boldsymbol{o}),$$

where $X_i \subseteq X$ denotes the set of instances with class labels y_i, and these class labels are assumed to have the order $y_1 < y_2 < \cdots < y_m$. The training data consists of a set of labeled instances, just like in classification. Since CLeRa is learning from pairwise comparisons of the form $(\boldsymbol{o}^*, \boldsymbol{o})$, it first extracts such comparisons from the original data by looking at the class information: A preference $(\boldsymbol{o}^*, \boldsymbol{o})$ is generated for each pair $(\boldsymbol{o}^*, y_j)$ and $(\boldsymbol{o}, y_i)$ of labeled instances in the (original) training data such that $y_i < y_j$.

The ranking performance of CLeRa (with maximum grouping size of $g_{max} = 2$) is compared with LexRank, which was implemented as proposed by Flach and Matsubara [12, 11]; therefore, this method was only applied to binary (two-class) problems but not to problems with more than two classes.[2] We applied naive Bayes (NB) and decision tree (J48) learning as additional baselines, using the standard implementations in Weka (trees are not pruned) and sorting instances according to the estimated probability of the positive class; note that these methods are not applicable to the multi-class case either.

Table 1. Average performance in terms of C-index based on a 10-fold cross-validation.

Dataset	CLeRa	LexRank	J48	NB
Red Wine	0.7827	0.8011	0.7378	0.8110
Census Income	0.7952	0.5776	0.7401	0.8607
Credit Approval	0.9201	0.9229	0.8517	0.9061
Mammographic Mass	0.8831	0.8960	0.8524	0.8999
Mushroom	1.000	0.9865	1.0000	0.9484
SPECT Heart	0.674	0.6590	0.5106	0.7409
Ionosphere	0.9198	0.5748	0.8059	0.9061
MAGIC Gamma Telescope	0.8218	0.7263	0.7841	0.8241
Breast Cancer Wisconsin	0.9837	0.9901	0.9793	0.9909
German Credit	0.6285	0.4523	0.6251	0.7835
Car Evaluation	0.9198	n/a	n/a	n/a
Nursery	0.9052	n/a	n/a	n/a
Tic-Tac-Toe Endgame	0.7728	n/a	n/a	n/a
Vehicle	0.7554	n/a	n/a	n/a
Cardiocraphic	0.9551	n/a	n/a	n/a

The results of a 10-fold cross-validation are given in Table 1. Since CLeRa produced a completeness of 1 or extremely close to 1 throughout, these values are not reported here. Overall, the performance of the methods is quite comparable. In particular, CLeRa and LexRank produce quite similar results on many data sets. In some cases, however, the results are strongly in favor of CLeRa (Census Income, Ionosphere, MAGIC Gamma Telescope, German Credit). Probably, this is because the bias imposed by the assumption of a standard lexicographic order is inadequate for these data sets, and hence our extensions (conditional attribute importance, conditional value preferences, variable grouping) clearly pay off.

6 CONCLUSIONS AND FUTURE WORK

Lexicographic orders constitute an interesting model class for preference learning, which allows for representing rankings of a set of objects in a very compact and comprehensible way. Yet, as we have argued in this paper, this model class may not be flexible enough for many real-world applications. Therefore, we have proposed to weaken the assumptions underlying a lexicographic order in various directions, allowing for conditional attribute importance, conditional preferences on attribute values, and variable grouping. Moreover, we have proposed an algorithm called CLeRa, which learns preference models in the form of conditional lexicographic preference trees from training data in the form of pairwise comparisons between objects.

First experimental results in the setting of multipartite ranking are quite promising and show CLeRa to be competitive with other methods. In a direct comparison with an existing lexicographic ranker, the benefit of our extensions are becoming quite obvious.

Important topics of future work can be found both on the theoretical and practical side. In particular, we are currently studying formal properties of our generalized model class, such as its expressiveness and means for regularization and complexity control. Practically, there is certainly scope for improving our current algorithm, for example by devising a suitable procedure for estimating an optimal value g_{max} for the oder of variable grouping. Moreover, improving the computational efficiency of CLeRa would be desirable, too. Last but not least, we are of course interested in real applications for which (generalized) lexicographic models appear to be an adequate representation.

[2] The red wine data actually has a target attribute with values between 1 and 10; it was binarized by thresholding at the median.

REFERENCES

[1] M. Ahlert, 'Aggregation of lexicographic orderings', *Homo Oeconomicus*, **25(3/4)**, 301–317, (2008).

[2] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[3] R. Booth, Y. Chevaleyre, J. Lang, J. Mengin, and C. Sombattheera, 'Learning various classes of models of lexicographic orderings', in *Workshop on Preferencce Learning at ECML-2009*, (2009).

[4] R. Booth, Y. Chevaleyre, J. Lang, J. Mengin, and C. Sombattheera, 'Learning conditionally lexicographic preference relations', in *Proc. ECAI 2010*, pp. 269–274, Amsterdam, The Netherlands, (2010). IOS Press.

[5] C. Boutilier, R.I. Brafman, C. Domshlak, H.H. Hoos, and D. Poole, 'CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements', *Journal of Artificial Intelligence*, **21**, 135–191, (2004).

[6] W. Cheng, M. Rademaker, B. De Baets, and E. Hüllermeier, 'Predicting partial orders: Ranking with abstention', in *Proc. ECML-2010*, (2010).

[7] William Cohen, Robert Schapire, and Yoram Singer, 'Learning to order things', *Journal of Artificial Intelligence Research*, **10**, 243–270, (1999).

[8] O. Dekel, C. Manning, and Y. Singer, 'Log-linear models for label ranking', in *Advances in Neural Information Processing Systems*, (2003).

[9] J. Dombi, C. Imreh, and N. Vincze, 'Learning lexicographic orders', *European Journal of Operational Research*, **183**(2), 748–756, (2007).

[10] P.C. Fishburn, 'Lexicographic orders, utilities and decision rules: A survey', *Management Science*, **20**(11), 1442–1471, (July 1974).

[11] P.A Flach and E.T. Matsubara, 'On classification, ranking and probability estimation', in *Probabilistic, Logical and Relational Learning - A Further Synthesis*, (2007).

[12] P.A. Flach and E.T. Matsubara, 'A simple lexicographic ranker and probability estimator', in *Proceedings of the 18th European conference on Machine Learning*, ECML '07, pp. 575–582, Berlin, Heidelberg, (2007). Springer-Verlag.

[13] J. Fürnkranz and E. Hüllermeier, *Preference Learning*, Springer-Verlag, Berlin, Heidelberg, 2010.

[14] J. Fürnkranz, E. Hüllermeier, and S. Vanderlooy, 'Binary decomposition methods for multipartite ranking', in *Proceedings ECML/PKDD–2009, European Conference on Machine Learning and Knowledge Discovery in Databases*, Bled, Slovenia, (2009).

[15] E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker, 'Label ranking by learning pairwise preferences', *Artificial Intelligence*, **172**, 1897–1917, (2008).

[16] M. Schmitt and L. Martignon, ‘On the complexity of learning lexicographic strategies’, *J. Mach. Learn. Res.*, **7**, 55–83, (December 2006).

[17] N. Wilson, ‘Efficient inference for expressive comparative preference languages’, in *Proc. IJCAI-09*, (2009).

[18] F. Yaman, T.J. Walsh, M.L. Littman, and M. desJardins, ‘Democratic approximation of lexicographic preference models’, in *Proc. ICML-08*, pp. 1200–1207, Helsinki, Finland, (2008).

An apple-to-apple comparison of Learning-to-rank algorithms in terms of Normalized Discounted Cumulative Gain

Róbert Busa-Fekete[1 2] and **György Szarvas**[3] and **Tamás Éltető** [4] and **Balázs Kégl** [5]

Abstract. The Normalized Discounted Cumulative Gain (NDCG) is a widely used evaluation metric for learning-to-rank (LTR) systems. NDCG is designed for ranking tasks with more than one relevance levels. There are many freely available, open source tools for computing the NDCG score for a ranked result list. Even though the definition of NDCG is unambiguous, the various tools can produce different scores for ranked lists with certain properties, deteriorating the empirical tests in many published papers and thereby making the comparison of empirical results published in different studies difficult to compare. In this study, first, we identify the major differences between the various publicly available NDCG evaluation tools. Second, based on a set of comparative experiments using a common benchmark dataset in LTR research and 6 different LTR algorithms, we demonstrate how these differences affect the overall performance of different algorithms and the final scores that are used to compare different systems.

1 Introduction

In *subset ranking* [3] (or *web page ranking*), the goal is to learn a ranking function that approximates the ideal partial ordering of a set of objects (or documents retrieved for the same query). The partial ordering is provided by relevance labels representing the relevance of documents with respect to the query on an absolute scale.

In the past, manually designed ranking functions, such as BM25 [7], were used to rank the retrieved documents in web page ranking. More recently, this problem is tackled as a machine learning task, where the training data is given in the form of (query, document, relevance label) triplets. These machine learning based ranking approaches are referred to as *learning-to-rank* (LTR) systems.

The Normalized Discounted Cumulative Gain (NDCG) is a widely used evaluation metric for learning-to-rank systems. NDCG is designed for ranking tasks with more than one relevance levels. There are many freely available, open source tools for computing the NDCG score for a ranked result list (for full list of these tools see Section 3). Even though the definition of NDCG is unambiguous[6], the various tools can produce different scores for ranked lists with certain properties, deteriorating the empirical tests in many published papers and thereby making the comparison of empirical results published in different studies difficult to compare.

We found that for certain benchmark datasets, the relative order of the performance of different LTR methods can change depending on which evaluation tool was used. The reason for this can be two-fold. First, the implemented NDCG calculation can itself result in different scores, and in some cases, a different order for the same ranked result lists. Second and more importantly, we found some of the LTR algorithms that compute the NDCG scores during the training phase to be sensitive to the different ways of computing the NDCG score, i.e. depending on which NDCG implementation is used, some LTR methods can produce different models that provide significantly different overall performance. In previous work[2], we recognized this, here we investigate the effect of using different evaluation tools more detailed.

The contribution of this study is two-fold. First, we identify the major differences between the various publicly available NDCG evaluation tools. Second, based on a set of comparative experiments using a common benchmark dataset in LTR research and 6 different LTR algorithms, we demonstrate how these differences affect the overall performance of different learning methods and the final scores that are used to compare different systems. We analyze these differences and also draw conclusions on which metric should be used and why.

This is not only an important step towards making the empirical research results reported in the literature uniformly comparable, but it also has implications on how benchmark datasets should be designed in order to be equally suited to train and evaluate any particular learning to rank algorithm.

The rest of this paper is organized as follows. In the next section we will describe the formal setup, and then, in Section 3 we list all the evaluation tools that are available freely to compute NDCG and, in addition, we identify precisely the differences in the way they compute the NDCG score. Next, in Section 4 we briefly overview the LTR algorithms we are used for comparison in the experiments in Section 5. Finally, based on the observed differences in experimental results, we draw our conclusions in Section 6.

[1] Department of Mathematics and Computer Science, Marburg University, Germany, email: busarobi@mathematik.uni-marburg.de

[2] R. Busa-Fekete is on leave from the Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged.

[3] Computer Science Department Technische Universität Darmstadt, Germany, email :szarvas@tk.informatik.tu-darmstadt.de

[4] Ericsson Hungary, Könyves Kálmán krt. 11.B, 1097 Budapest, Hungary, email: tamas.elteto@ericsson.com

[5] Linear Accelerator Laboratory (LAL) and Computer Science Laboratory (LRI), University of Paris-Sud, CNRS, Orsay, 91898, France, email: balazs.kegl@gmail.com

[6] See, for example `http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html`

2 Formal LTR task

In this section we formally define the learning-to-rank problem and introduce the notation that will be used in the rest of the paper. Let us assume that we are given a set of *query objects* $\mathbf{D} = \{\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(M)}\}$. Each query object $\mathcal{D}^{(k)}$ consists of a set of $n^{(k)}$ pairs

$$\mathcal{D}^{(k)} = \Big\{ \big(\mathbf{x}_1^{(k)}, \ell_1^{(k)}\big), \ldots, \big(\mathbf{x}_{n^{(k)}}^{(k)}, \ell_{n^{(k)}}^{(k)}\big) \Big\}.$$

The real-valued feature vectors $\mathbf{x}_i^{(k)}$ represent the kth query and the ith document received as a potential hit for the query.[7] The *label index* $\ell_i^{(k)}$ of the query-document pair $\mathbf{x}_i^{(k)}$ is an integer between 1 and K. We assume that we are given a set of numerical *relevance grades*

$$\mathcal{Z} = \{z_1, \ldots, z_K\}.$$

The relevance grade $z_i^{(k)} = z_{\ell_i^{(k)}}$ expresses the relevance of the ith document to the kth query on a numerical scale. A popular choice for the numerical relevance grades is

$$z_\ell = 2^{\ell-1} - 1 \tag{1}$$

for all $\ell = 1, \ldots, K$.

The goal of the ranker is to output a permutation $\mathbf{j}^{(k)} = (j_1, \ldots, j_{n^{(k)}})$ over the integers $(1, \ldots, n^{(k)})$ for each query object $\mathcal{D}^{(k)}$. A widely used empirical measure of the quality of the permutation $\mathbf{j}^{(k)}$ is the Discounted Cumulative Gain (DCG)

$$\mathrm{DCG}\big(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}\big) = \sum_{i=1}^{n^{(k)}} c_i z_{j_i}^{(k)}, \tag{2}$$

where c_i is the *discount factor* of the i^{th} document in the permutation. The most common discount factor is

$$c_i = \frac{1}{\log(1+i)}. \tag{3}$$

The rationale of this formula is that a user will be particularly satisfied if he/she finds relevant documents early in the permutation. To normalize DCG between 0 and 1, (2) is usually divided with the DCG score of the best permutation (NDCG) which can be computed as

$$\mathrm{IDCG}\big(\mathcal{D}^{(k)}\big) = \max_{\mathbf{j}} \mathrm{DCG}\big(\mathbf{j}, \mathcal{D}^{(k)}\big)$$

This score referred to as the *ideal* DCG score. It is also a common practice to truncate the sum (2) at $n_{\max}$, defining the $\mathrm{DCG}_{n_{\max}}$ and $\mathrm{NDCG}_{n_{\max}}$ scores. The reason for this is that a user would rarely browse beyond the first page of search results containing the first $n_{\max}$ hits.

3 Evaluation tools

In this subsection we briefly describe and compare the various tools available to compute NDCG scores. The definition (2) is unambiguous, nevertheless, the tools can differ in the definition of the discount factor c_i (3). More importantly, there can be an important difference in the way the DCG score is normalized when i) there is no relevant document for a query ($z_i = 0$ for all i), or ii) when the number of documents is less then the truncation level $n_{\max}$. Although this seems to be a technical subtlety, it turns out that the confusion between the different tools can significantly alter the numerical scores and in some case can even change the relative ordering of the algorithms on benchmark datasets.

We compared six evaluation tools computing the NDCG scores:

1. The LETOR 3.0 script implemented in Perl[8]
2. The LETOR 4.0 script implemented in Perl[9]
3. The MS script implemented in Perl[10]
4. The YAHOO script implemented in Python[11]
5. The RANKLIB package implemented in Java[12]
6. The TREC evaluation tool v8.1 implemented in C[13]

The evaluation tools can be divided into three groups. The tools of the first group compute $\mathrm{DCG}_{n_{\max}}$ according to the definition (2) described in Section 2. The LETOR 3.0, YAHOO, and TREC tools belong to this group. All of these tools assign zero score to a query if it is *empty*, that is, $z_i = 0$ for all i which means that there are no relevant documents. The TREC tool uses the labels of documents given in the input file as relevance grades by default. From this point of view, this is the most flexible implementation, since arbitrary relevance grades can be defined. For example, in the case of MQ2008 dataset, the labels 0, 1 and 2 should be simply replaced by 0, 1 and 3 respectively, to have the commonly used exponential grades (1).

The second group is composed of the YAHOO tool alone. It also computes the $\mathrm{DCG}_{n_{\max}}$ according to the definition (2), but it assigns 1.0 to the empty queries. This is a minor difference that generates an additive bias between the $\mathrm{NDCG}_{n_{\max}}$ computed by YAHOO tool and the three tools of the first group.

The third group consists of the LETOR 4.0 and MS tools. Except for a small technical difference (the LETOR 4.0 tool can be used for up to three relevance labels, whereas the MS tool can handle up to five relevance labels), they compute the same score. As the RANKLIB and LETOR 3.0 tools, they assign zero to a query where the ideal $\mathrm{DCG}_{n_{\max}}$ is zero. Their rather strange feature is that they also assign zero $\mathrm{DCG}_{n_{\max}}$ score to a query with less then $n_{\max}$ documents in it, even if these documents are highly relevant. So, formally, they compute the $\mathrm{DCG}_{n_{\max}}$ score as

$$\mathrm{DCG}_{n_{\max}}\big(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}\big) = \begin{cases} \sum_{i=1}^{n_{\max}} c_i z_{j_i}^{(k)} & \text{if } n_{\max} \le n^{(k)} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

This truncation does not only distort the *test* score, but it can also alter the *training* of such algorithms that depend directly on the NDCG score. Indeed, for example, in ADARANK [9] which optimizes the NDCG_{10} evaluation metric, a query containing less than 10 documents does not influence the computation of the coefficient of the weak ranker at all, and the the weight of such queries converge to zero over the boosting iterations.

The only source of difference between the tools that have not been identified yet, is the way they sort the objects of interest based on the scores. All tools except TREC tool, make use of the default built-in, programing language dependent sorting function which mainly

[7] When it is not confusing, we will omit the query index and simply write $\mathbf{x}_i$ for the ith document of a given query.

[8] http://research.microsoft.com/en-us/um/beijing/projects/letor/LETOR3.0/EvaluationTool.zip

[9] http://research.microsoft.com/en-us/um/beijing/projects/letor/LETOR4.0/Evaluation/Eval-Score-4.0.pl.txt

[10] http://research.microsoft.com/en-us/projects/mslr/eval-score-mslr.pl.txt

[11] http://learningtorankchallenge.yahoo.com/evaluate.py.txt

[12] http://www.cs.umass.edu/~vdang/ranklib.html

[13] http://trec.nist.gov/trec_eval/

implement the quick sort algorithm. Therefore, the order of two elements with the same score depends on the implementation of the sorting algorithm used. Whereas the TREC tool uses the lexicographic ordering based on document ID given in the input file if the system-predicted scores are equal for two documents.

We believe that none of these two ways of handling equal scores are desirable. On the one hand, the sorting algorithm should not have an effect on the evaluation itself. On the other hand, the document ID normally does not bear any useful information regarding the content of a document, so in this sense, the use of the ordering based on document ID corresponds to a particular random ordering which is also not a desired feature in an evaluation tool.

4 Learning-to-rank algorithms

In our comparison study, we used six state-of-the-art ranking methods. Here we briefly summarize them.

1. ADARANK [9] is a listwise boosting approach aiming to optimize an arbitrary listwise IR metrics, such as the Mean Average Precision (MAP), ERR, or NDCG. Inspired by ADABOOST, it uses a stepwise greedy optimization technique for maximizing the chosen IR metrics. In every boosting iteration, ADARANK reweights the queries based on their scores obtained by the evaluation metrics: it up-weights the query having lower score and down-weights high-scoring queries. The weak learner is chosen by optimizing the listwise evaluation metrics of interest which is usually hard to optimize except for very simple weak classifiers. This can be viewed as a handicap of this method. According to the original implementation of ADARANK, we used the best feature ranker (BF) described above as base ranker taking into account the weighting of queries. The only hyperparameter of ADARANK is the number of boosting iterations which we optimized by using early-stopping on the validation set. We refer to this method as ADARANK.$\{\mathrm{NDCG}\}$.
2. RANKNET [1] is a neural-net-based method which employs a loss based on pairwise cross entropy as its objective function. The neural net with one output node is trained to optimize directly the differentiable probabilistic pairwise loss instead of the common squared loss. We validated the number of hidden layers ranging from 1 to 3 and the number of neurons in the hidden layers ranging from 10 to 500. For the number of training epochs we applied early stopping.
3. RANKBOOST [4] is a pairwise boosting approach. The objective function is the rank loss (as opposed to ADABOOST which optimizes the exponential loss). In each boosting iteration the weak classifier is chosen by maximizing the weighted rank loss. For the weak learner we used decision stumps and a variant of the single decision stump described in [4] which is able to optimize the rank loss in an efficient way.
4. RANKSVM [5] is a pairwise method based on SVM, formulating the ranking task as a binary classification. We used linear kernel because the optimization using non-linear kernels cannot be carried out in reasonable time. The tolerance level of the optimization was set to 0.001 and the regularization parameter was validated in the interval $[10^{-6}, 10^4]$ with a logarithmically increasing step
5. COORDINATEASCENT (CA) [6] is a linear listwise model where the scores of the query-document pairs are calculated as weighted combinations of the feature values. The weights are tuned by using a coordinate ascent optimization method where the objective function is an arbitrary evaluation metrics given by the user. The coordinate ascent optimization method itself has two hyperparameters to be tuned: the number of restarts R from random initial weights, and the number of iterations T taken after each restart. We used $R = 30$ and $T = 100$. We did not validate these hyperparameters, but using the validation set we evaluated every model obtained due to restarting the optimization process, and we kept the one having highest performance.
6. LAMBDAMART [8] is a boosted regression tree model. Since it handles the LTR problem as a regression task, it could be classified as pointwise method, but during the training phase, it adjust the parameters of the regression trees based on the derivative estimate of NDCG, therefore it is considered as a listwise approach. We validated the number of boosting iterations. The number of leaves were set to 10 and the learning rate to 0.1.

5 Experiments

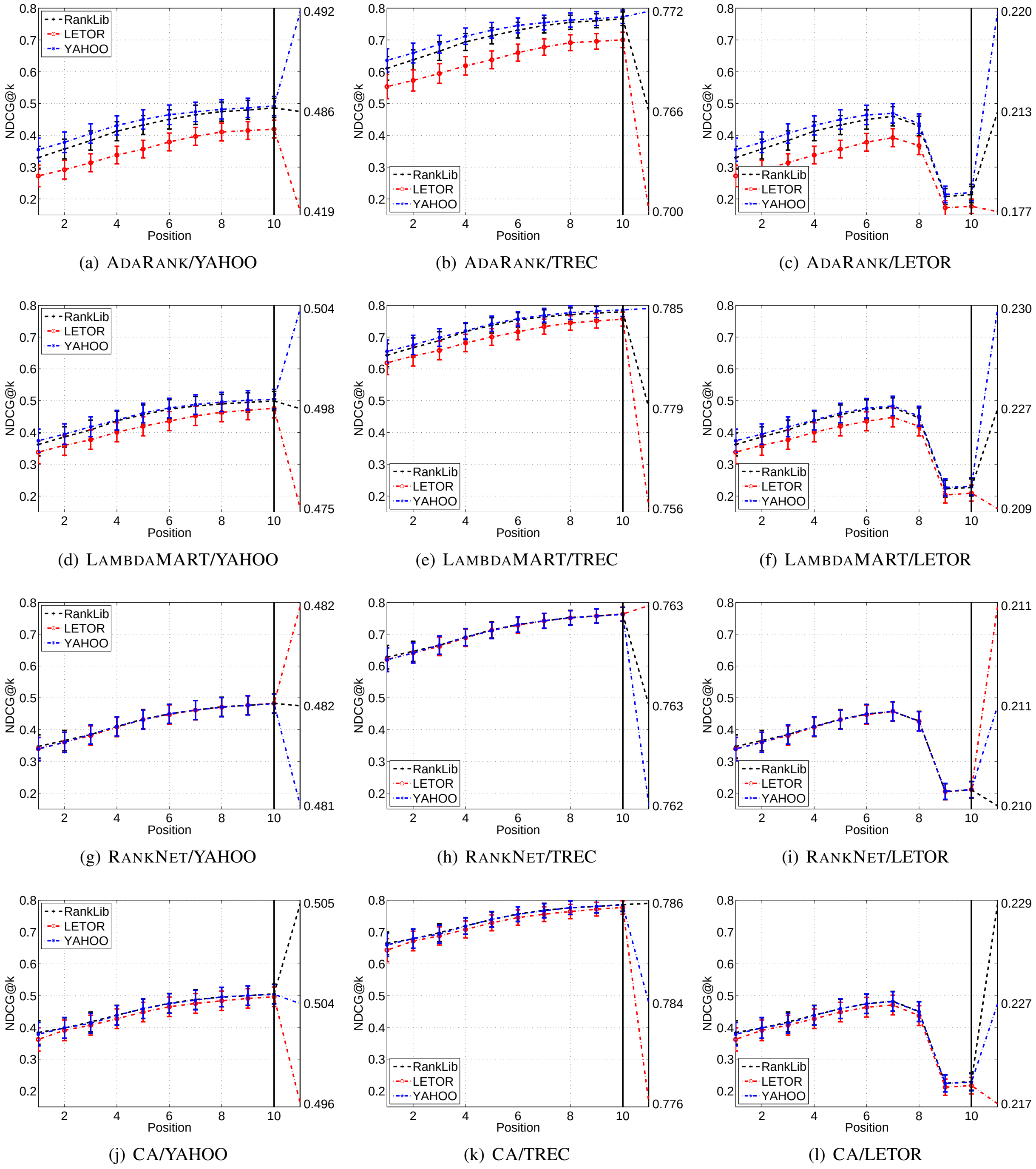
We identified two major differences in the way the DCG score is normalized in publicly available NDCG evaluation scripts:

1. When there is no relevant document for a query ($z_i = 0$ for all i), evaluation scripts conforming to the definition assign a 0.0 NDCG value to the query. Optionally, some scripts may assign a different value as default for such queries (namely, 1.0 for the Yahoo metric).
2. When the number of documents is less then the truncation level $n_{\max}$, evaluation scripts conforming to the definition assign an $\mathrm{NDCG}_{n^{(k)}}$ value equal to the number of documents $n^{(k)}$ available for that query (thereby assuming that it is always possible to fill in a result list with irrelevant documents, and at the same time assuming that the provided relevance labeling is exhaustive (i.e. there are no further, unseen relevant documents and thus the use of $\mathrm{IDCG}_{n^{(k)}}$ score in the normalization step is a realistic estimate.[14]) Optionally, some scripts may assign a different value as default for such queries (namely, 0.0 for the LETOR metric).

We consider the difference stemming from the different strategy to order results with equal predicted scores less crucial (in practice two different documents seldom get the same predicted score, i.e. ties are rare) and we do not assess its impact, and we use all metrics with exponential relevance grades (the TREC tool can be parameterized to use exponential gain, while the other tools are implemented to use this). That is, in our experiments we consider the Ranklib and TREC evaluation tools to be equivalent and compare their scores to those provided by the YAHOO and LETOR metrics, which are representative examples for the two major differences we found between the various tools.

In Figure 1, we plot the NDCG_{1-10} scores for 4 characteristic learning to rank algorithms (ADARANK, LAMBDAMART, RANKNET and CA) using the three different evaluation formulas to evaluate the models: in each row, the left, middle and right plots show the values provided by the RANKLIB/TREC, the YAHOO and the LETOR tools, respectively. All models were trained to optimize the NDCG_{10} scores, and the plots show the NDCG scores of these models for cut-off values from 1 to 10. Each plot shows 3 different curves that correspond to models trained using a specific metric during the training of the model: the blue, black and red lines indicate models learned using the YAHOO, RANKLIB/TREC and LETOR metrics, respectively. This way we can visually compare the effects

[14] Note that relevance labels are many times pooled in IR datasets and there is no guarantee that no relevant, but unlabeled documents exist.



(a) AdaRank/YAHOO

(b) AdaRank/TREC

(c) AdaRank/LETOR

(d) LambdaMART/YAHOO

(e) LambdaMART/TREC

(f) LambdaMART/LETOR

(g) RankNet/YAHOO

(h) RankNet/TREC

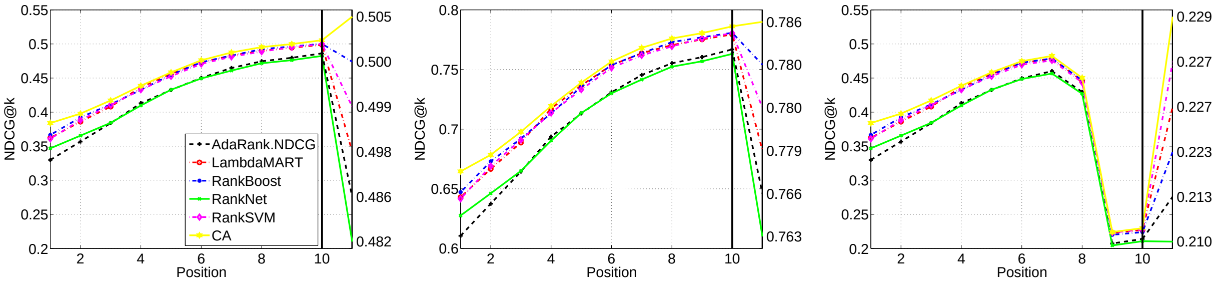
(i) RankNet/LETOR

(j) CA/YAHOO

(k) CA/TREC

(l) CA/LETOR

Figure 1. The dependence of NDCG_{10} scores on NDCG method used on MQ2008 dataset.



(a) YAHOO

(b) TREC

(c) LETOR

Figure 2. The dependence of NDCG_{10} scores on NDCG method used on MQ2008 dataset.

of the different evaluation tools both on on the numeric outcomes (comparing the curves with same color, denoting the same models, across the 3 plots horizontally) and on the learnt model (contrasting the 3 curves on the same plots, denoting models learnt using different measures).

We can make several observations based on figure1. First, for all the algorithms in figure 1, the NDCG scores provided by the YAHOO tool are on average around 0.25 higher than the scores provided by the RANKLIB/TREC tool which conforms the definition of NDCG. This difference comes solely from the presence of queries with 0 relevant documents in the dataset: for those queries, the YAHOO tool assign 1.0 as NDCG which results in an additive bias compared to the official NDCG scores. On the other hand, the scores provided by the LETOR tool degrade rapidly for cutoff values of $8 - -10$ for all algorithms, with a difference around or above 0.25 compared to the official NDCG scores. As opposed to the former additive difference this behavior might result in a different ordering of the algorithms: a method that is on average superior to others on larger sets but is weaker than the competitors on small sets is preferred by the Letor metric, as this tool assigns zero to small sets (smaller than the NDCG cut-off).

Second, and more interestingly, we can observe that it is by no means irrelevant which metric was used during the training of the algorithms: for those algorithms that make use of the NDCG scores in some way in the training process (namely, the ADARANK, LAMBDAMART and CA methods), we see differences in the performance of the learnt models. We observe that in general, the LETOR tool is not suited for training the algorithms – its property to assign 0.0 score to small sets causes these small sets to be useless for training (to optimize NDCG_{10}), i.e. the algorithms can exploit less data in a meaningful way to learn patterns. This results in a significantly[15] worse performance for these algorithms, when trained using the LETOR metric (instead of the official NDCG scores). On the other hand, for some algorithms, it is worth to assign a non-zero score to sets which contain no relevant documents at all: doing so, the algorithms do not increase the weight of such queries similarly to other low-performing queries where there is hope to improve performance. This can result in better learning rates and in some cases slightly better performance (see e.g. the blue vs. black curves for the ADARANK and LAMBDAMART methods).

Overall, we can observe that the best metric to train the algorithms is the one provided by YAHOO: in general this results in equal or better learnt models than the other evaluation tools. To shed light on how these characteristics of the evaluation tools affect the relative performance of benchmark LTR algorithms, in figure 2, we plot the NDCG scores for 6 different learning algorithms (trained using the official NDCG metric) with all the evaluation tools surveyed in this study. As can be seen, the performance of the best algorithms is very close to each other, with Coordinate Ascent having a slight advantage regardless which metric is used for evaluation. Slightly worse than CA, the three algorithms RANKBOOST, RANKSVM and LAMBDAMART perform very close to each other. If we compare the results obtained with different evaluation measures, the relative order of these (otherwise very similar) models change depending on the metric: apparently the RANKBOOST algorithm has a slight advantage over the other two for short sets, which advantage disappears when the LETOR metric is used for evaluation. In general, we were unable to reproduce the competitive results of ADARANK [9], using the reimplementation of the algorithm in the Ranklib package [16],[17].

6 Conclusion

In this study, we reviewed the publicly available NDCG evaluation scripts, identified and compared the differences between them and systematically analyzed how these differences affect the numeric results and the training processes of various learning-to-rank algorithms. It is reasonable to assume that most previous studies use one of the assessed evaluation tools, and at the same time it seems likely that most measures are used at least in some studies to evaluate the performance of machine learnt ranking systems. We found that there indeed are differences between tools despite the relative simplicity of the popular NDCG evaluation metric, and our experiments demonstrate that these differences can easily lead to non-trivial differences between research results. Since most studies do not discuss such details as the evaluation script used, this fact makes previous studies very difficult to compare and might lead to the misinterpretation of results and false conclusions.

We identified that the two key points of difference between different tools are i) the way how small queries (queries with less than $n_{\max}$ documents in the case of $\text{NDCG}_{n_{\max}}$) and ii) the way how queries with no relevant document (i.e. when all documents have the same, zero relevance score and therefore the ideal DCG is zero) are handled by the tools. These two factors can lead to different overall scores (for the same model) and also to inherently different learnt models, depending on which learning algorithm was used. Some divergences from the definition of the NDCG measure might be jus-

[15] The error bars on the plots correspond to the standard errors of querywise NDCG scores averaged out in quadrature over the folds.

[16] `http://people.cs.umass.edu/~vdang/ranklib.html`

[17] We found no clear indication in the original article what stopping criterion was used to terminate the iteration in the training process, so we tried to use various stopping strategies and provide the best results we obtained. These are nevertheless lower than those reported at the letor website.

tified from an ML perspective though: for example, if the measure assigns a perfect score to queries with zero relevant documents, these queries are not treated like other queries that can be improved, and do not distort the model. On the other hand, we could not identify any benefit of other modifications, such as zeroing out the $\mathrm{NDCG}_{n_{\max}}$ score for queries with less than $n_{\max}$ documents in total.

To summarize our findings, we suggest the following protocols to make learning to rank results more comparable and benchmark datasets more suited to training and evaluating systems:

1. The use of tools that give zero score to small queries should be avoided, as these can negatively impact certain algorithms (while others are unaffected) and thus the reported results can represent a false relative order of the algorithms.
2. If possible, benchmark datasets should be free of not meaningful queries. Queries that does not have any relevant documents do not play a role in learning to rank – they cannot be used to learn meaningful patterns and they do not have an impact on evaluation. At the same time, such queries can negatively impact the performance of some algorithms, and can motivate non-standard evaluation tools (c.f. the YAHOO metric).
3. Regardless which evaluation script is used while training the systems (c.f. the YAHOO tool which is reasonable in the presence of queries without relevant documents in the data), the final evaluation should be carried out using a tool fully conforming the official NDCG definition. This way machine learnt performance scores would become directly comparable to non-machine learnt ones, such as those coming from TREC and other evaluation exercises.

REFERENCES

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, ‘Learning to rank using gradient descent’, in *Proceedings of the 22th International Conference on Machine Learning*, (2005).

[2] R. Busa-Fekete, B. Kégl, T. Éltető, and Gy. Szarvas, ‘Tune and mix: learning to rank using ensembles of calibrated multi-class classifiers’, *Machine Learning* accepted, (2012).

[3] D. Cossock and T. Zhang, ‘Statistical analysis of Bayes optimal subset ranking’, *IEEE Transactions on Information Theory*, **54**(11), 5140–5154, (2008).

[4] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, ‘An efficient boosting algorithm for combining preferences’, *Journal of Machine Learning Research*, **4**, 933–969, (2003).

[5] R. Herbrich, T. Graepel, and K. Obermayer, ‘Large margin rank boundaries for ordinal regression’, in *Advances in Large Margin Classifiers*, eds., Smola, Bartlett, Schoelkopf, and Schuurmans, pp. 115–132. MIT Press, Cambridge, MA, (2000).

[6] D. Metzler and B. W. Croft, ‘Linear feature-based models for information retrieval’, *Information Retrieval*, **10**, 257–274, (2007).

[7] S. Robertson and H. Zaragoza, ‘The probabilistic relevance framework: BM25 and beyond’, *Found. Trends Inf. Retr.*, **3**, 333–389, (2009).

[8] Q. Wu, C.J.C. Burges, K.M. Svore, and J. Gao, ‘Adapting boosting for information retrieval measures’, *Information Retrieval*, **13**(3), 254–270, (2010).

[9] J. Xu and H. Li, ‘AdaRank: a boosting algorithm for information retrieval’, in *SIGIR ’07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 391–398. ACM, (2007).

Alleviating cold-user start problem with users' social network data in recommendation systems

Eduardo Castillejo and **Aitor Almeida** and **Diego López-de-Ipiña** [1]

Abstract. The Internet and the Web 2.0 have radically changed the way of purchasing items, provoking the fall of geographic selling barriers all over the world. So large is the amount of data and items we can find in the Web that it turned out to be almost unmanageable. Due to this situation many algorithms have emerged trying to filter items for e-commerce users based in their tastes. In order to do this, these systems need information about the tastes of the users as input. This limitation is reduced as the users interaction with these systems increases. The main problem arises when new users enter a recommendation platform for the first time. The so called cold-start problem causes unsatisfactory random recommendations, which goes against these systems' purpose. Cold-start includes users entering new systems, items, and even new systems. This situation challenges for new ways of obtaining user data. Social networks can be seen as huge information databases sources, and social network analysis would help us to do it using different techniques. In this paper, we present a solution which uses social network user data to generate first recommendations, alleviating the cold-user limitation. Besides, we have demonstrate that it is possible to reduce the cold-user problem applying our solution in a recommendation system environment.

1 INTRODUCTION

The amount of information in the world is increasing far more quickly than our ability to process it [17]. Common users of Web based systems usually have to deal with such amount of data that their interaction can become slow, ending in serious loss of users' attention, which also means losses of sells and user satisfaction. For example, buying a CD or a vinyl in a music store has been an habit from the 70s and 80s. People used to go to the store and navigate through tens or hundreds of albums seeking those which fit with their tastes. But nowadays, with all possibilities Internet offers to every user this amount of items has been increased to millions, even more.

Therefore, taking advantage of the possibilities the Web 2.0 provides to us all, researches started to design algorithms which were able to filter the information (or items) to the user. These algorithms started to compound what we today know as recommender systems. These systems use the opinions of a community of users to help individuals in that community to more effectively identify content of interest from a potentially overwhelming set of choices [16]. Within past years there have been many progresses in this area [4]. Some systems, such as YouTube, started to store some information about users' searches to infer their tastes [8, 5] managing some explicit and implicit information about users interactions. Others, such as Amazon.com[2], take into account the users' ratings and purchases [12]. Both points of view are different sides of the same coin and follow the same purpose: to present to the user the most suitable amount of items.

Despite the advances in this area there are some intrinsic problems that are still unsolved. Probably the most important one is the so called cold-user problem. It emerges every time a new user interacts with recommender system by the first time. Without any search, rating or purchase a recommender system is unable to find any interesting items. Sometimes it even has been necessary many ratings before being able to provide a reasonable recommendation [2].

Given the problem we asked ourselves a simple question: Is there any other way to infer user tastes without their active participation in the system? We think there is, and in this work we pretend to reduce the cited problem taking into account users' social interactions.

Social networks strength lies in the possibility of establishing some social relationships among people, companies and other groups using the Internet as the bridge of communication. The range of accessible social networks is very diverse, from those which just take into account the user location in order to rate a place, bar or store, to those which allows users to share not only their thoughts or beliefs, even their personal pictures, videos and music. There is such amount of information of the users in these networks that new challenges arise to take advantage of it.

This way, our proposal seizes the opportunity of exploiting social network data in order to reduce the cold-start problem in any recommendation system. Applying social network analysis techniques we are able to get some recommendations to the user (with certain accuracy) based on the relationships with others in social networks.

The remainder of this paper is structured as follows: first, in Section 2 we analyse the current state of the art in recommendation systems and the most popular techniques to avoid the cold-user or cold-start problem. Next we present our methodology for collecting valuable data from social networks taking into account users' relationships (Section 3). In Section 4 we analyse the results obtained from our proposal. Finally, in Section 5, we summarize our experiences and discuss the conclusions and future work.

2 RELATED WORK

Since the mid-1990s recommender systems have become an important research area attracting the attention of e-commerce companies. Amazon [12], Netflix[3] and Yahoo! Music[4] [6] are widespread exam-

[1] Deusto Institute of Technology - DeustoTech, University of Deusto, Avda. Universidades 24, 48007 - Bilbao, Spain. email: {eduardo.castillejo, aitor.almeida, dipina}@deusto.es

[2] http://www.amazon.com

[3] http://www.netflixprize.com/

[4] http://music.yahoo.com/

Table 1: Some of the best known metrics in social network analysis.

Metric	Description
Betweenness	It takes into account the connectivity of the node's neighbours to reflect the number. of people who a person is connecting indirectly through their direct links.
Centrality	It gives a rough indication of the social power of a node based on how well they ”connect” the network.
Closeness	It reflects the ability to access information through the “grapevine” of network members.
Cohesion	It measures the degree to which actors are connected directly to each other by cohesive bonds.
Degree	The count of the number of ties to other actors in the network.
Eigenvector centrality	A measure of the importance of a node in a network.

ples on making recommendations to its users based on their tastes and previous purchases. Although these systems have evolved becoming more accurate, the main problem is still out there: to estimate the rating of an item which has not been seen by users. This estimation is usually based on the rest of items rated by the current user or on the ratings given by others where the rating pattern is similar to the user's one. Although there are different kinds of recommendation systems (content-based, collaborative filtering and hybrid techniques) [4] they all suffer from the same main limitations: sparsity and scalability [17] and cold-start problems [15].

Some authors have tried to avoid the problem of cold-start users by asking them a series of questions about their tastes or by proposing some studied items in order to get any rating [13, 7]. As we presume these solutions can usually cause displeasure on the users, becoming tedious and cumbersome activities. An-Te Nguyen et al. [4] have tried to reduce the cold-user problem by exploiting available data (e.g. age, occupation, location, etc.). In [18] authors present a metric (the CROC curve) to improve the evaluation of a recommender system performance.

Moreover, some authors have improved their recommendation algorithms combining users' social data from social networks [10, 9]. Although they don't tackle the cold-start problem, their idea of using the available social information of users as an input represents a new starting point for these systems (e.g. Foursquare[5] adds information about user geolocation). There is also an open research in the Carnegie Mellon University's School of Computer Science about how do people really inhabit their cities based on Foursquare data. The project groups check-ins by physical proximity and it measures “social proximity” by how often different people check in similar places. This way resulting areas are dubbed [3].

But in a social network there is more beyond users, relationships and data. Social network analysis (SNA) refers to methods used to analyse social networks, social structures made up of individuals called ”nodes”, which are connected by different representations of relationships (e.g. friendship, kinship, financial exchange, etc.). Figure 1 represents an example of a social network graph in which different nodes are connected each other by relation lines called “links”. Once we have empirical data on a social network new questions arise: Which nodes are the most central members? Which are the most peripheral? Which people are influenced by others? Which connections are most crucial? These questions and their answers represent the basic domain of SNA [14]. There are many metrics which measure different aspects in a social network taking into account the nodes and their edges. Table 1 shows some of the best known metrics in SNA. As depicted in Section 3 we have chosen the eigenvector centrality metric to face up to the cold-start problem. A variant of eigenvector centrality is used by Google search engine to rank Web pages [11], but it is based in the premise that the system already has data from the user to work with.

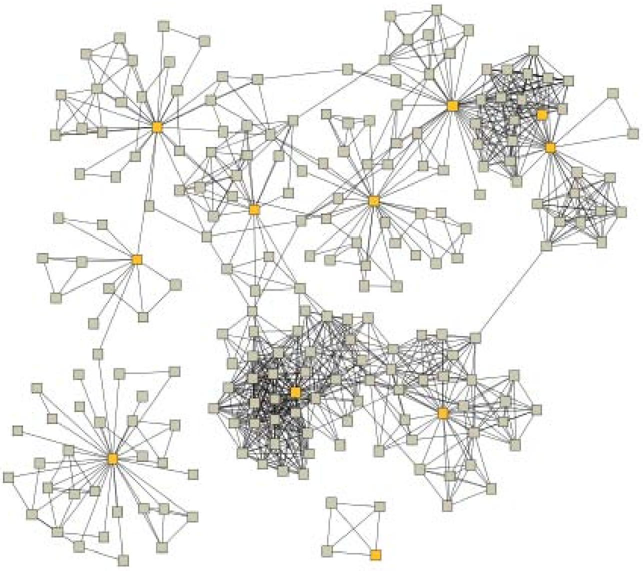


Figure 1: An example of a collaborative social network. Squares represent nodes (people) and the edges represent social ties between them. Yellow squares represent important nodes which relate different sub-graphs [14].

3 PROPOSED SOLUTION

This section details the developed system to enable the generation of generic recommendations to the user based in the rest of the users who checked in the same venue with Foursquare. To get the rest of the users (the network nodes) who checked in the same place we have used the Foursquare API. Once we have the nodes we calculate those which are the most important in the network at the current venue and then we obtain the recommendations which fit better to the user linking using probabilistic.

3.1 Foursquare API

Foursquare is a location-based social networking website which allows users to ”check in” at venues using their smartphones. The Foursquare API[6] gives access to all of the data used by the Foursquare mobile applications. Thanks to it developers can request some user data (e.g. location, friends, last check-ins, etc.). We have developed an Android mobile application which allows users to check in desired venues as they would do with Foursquare official application or website (see Figure 3). Once the user checks in any venue, our algorithm is launched: First, we must authenticate the user with the Oauth protocol (required by the Foursquare API). Then we are able to get the nodes who have checked in the current

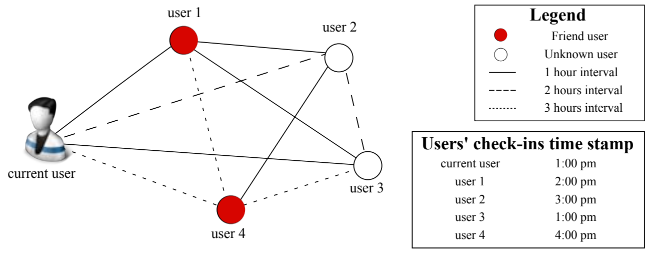
[5] http://www.foursquare.com

[6] https://developer.foursquare.com/

venue. To do this it is necessary to use the "herenow"[7] API endpoint aspect, which responses a count and items (where items are Checkin[8] responses) in JSON format. For example, doing a check-in at the Colosseo, in Rome, we obtained a JSON "herenow" response containing 4 items representing 4 different people who had previously checked in the same venue (see Figure 2). That's the point of the "herenow" endpoint, to get the previous check-ins done by other users at the current checked in venue.

Therefore, we can build a 5 × 5 square adjacency matrix (3) representing the network graph at the current venue for each user (the first column relates the current user with the others). Each A_{ij} element will be tied to another with a default weight value of 1. Then we start looking to the "createdAt" field of every Checkin object in the "herenow" JSON response. This field is a long number, and it's value is based in the the Unix epoch time (it stores the number of seconds that have elapsed since midnight Coordinated Universal Time (UTC), January 1, 1970). We have split the importance of every check-in in 3 time intervals, adding weights to the values in the matrix in the corresponding A_{ij} position. We decided this because a Foursquare check-in has a lifetime of 3 hours approximately, and that is the reason why we have chosen a 3 hours time interval to give weights to the check-ins. We also believe that people are defined by their acts, and this is why we defend the temporal closeness approach. Being at same places at similar time periods can be used as a tool for relate different people with some and probable similar tastes.

To complete the weights assignation we must take into account that the "herenow" request returns not only the list of the people who have checked in the same venue but also a list of our Foursquare friends who have also done it. Accordingly to this, we believe that our friends have an extra weight (or importance) in the network. The following (1) and (2) equations detail the possible weights distribution when a user checks in a venue. Given priority to user's friends we can see how the maximum weight for an unknown user is 3, although for a friend scales up to 6. We have implemented such algorithm because being friends in a social network does not directly imply that users' tastes are the same.



Users' check-ins time stamp	
current user	1:00 pm
user 1	2:00 pm
user 2	3:00 pm
user 3	1:00 pm
user 4	4:00 pm

Figure 2: Example of a graph built from a user check-in where there are 4 more Foursquare users who had checked in the same venue. Matrix A'' (4) represents the same graph.

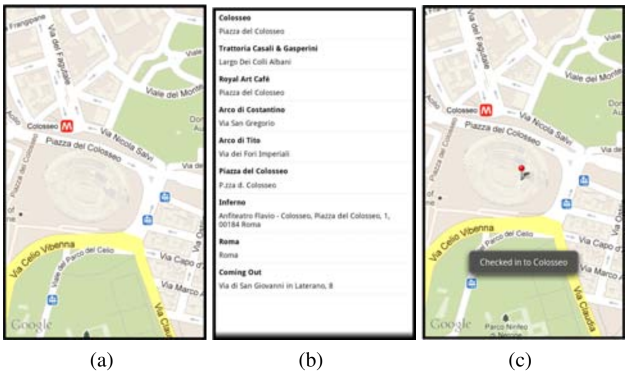
$$Unknown\_user = \left\{ \begin{array}{cc} 1 & \text{default weight} \\ +3 & \text{if check-in interval} \leq 1 \text{ hour} \\ +2 & \text{if 1 hour} < \text{check-in interval} \leq 2 \text{ hour} \\ +1 & \text{if 2 hour} < \text{check-in interval} \leq 3 \text{ hour} \end{array} \right\} \quad (1)$$

$$Friend\_user = \left\{ \begin{array}{cc} 3 & \text{default weight} \\ +3 & \text{if check-in interval} \leq 1 \text{ hour} \\ +2 & \text{if 1 hour} < \text{check-in interval} \leq 2 \text{ hour} \\ +1 & \text{if 2 hour} < \text{check-in interval} \leq 3 \text{ hour} \end{array} \right\} \quad (2)$$

The following matrices show how the first default values are added just when the adjacency matrix A is built and then how it is completed with the weights assignations (A''). A' is built by adding to A the corresponding values of the relationships between the current user and the others (see Figure 2 and equations (1) and (2)). Regarding at node A''_{01} we can see that it has a value of 6. This is because of the combination of the relationship between the current user and the "user 1" (they are friends) and the check-in time interval (1 hour).

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} \quad (3)$$

$$A' = \begin{pmatrix} 0 & 3 & 1 & 1 & 3 \\ 3 & 0 & 1 & 1 & 3 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{pmatrix} A'' = \begin{pmatrix} 0 & 6 & 3 & 4 & 4 \\ 6 & 0 & 3 & 3 & 1 \\ 3 & 3 & 0 & 2 & 3 \\ 4 & 3 & 2 & 0 & 1 \\ 4 & 1 & 3 & 1 & 0 \end{pmatrix} \quad (4)$$



Colosseo
Piazza del Colosseo
Trattoria Casali & Gasperini
Largo Dei Colli Albani
Royal Art Café
Piazza del Colosseo
Arco di Costantino
Via San Gregorio
Arco di Tito
Via dei Fori Imperiali
Piazza del Colosseo
P.zza d. Colosseo
Inferno
Anfiteatro Flavio - Colosseo, Piazza del Colosseo, 1, 00184 Roma
Roma
Roma
Coming Out
Via di San Giovanni in Laterano, 8

Checked in to Colosseo

(a) (b) (c)

Figure 3: A user checks in a venue with developed Android application. First a map is shown (a). Once the user touches the screen a near venues list is presented (b). Finally, when a venue is selected, the check-in is performed (c).

3.2 Eigenvector centrality

As we have depicted in Section 2 a network is a graph made up of points, nodes or vertices tied each other by edges. We can also represent these graphs by a so called adjacency symmetric $(n \times n)$ matrix, where n is the number of nodes. This matrix has elements

$$A_{ij} = \left\{ \begin{array}{cc} 1 & \text{if there is an edge between vertices } i \text{ and } j \\ 0 & \text{otherwise} \end{array} \right\} \quad (5)$$

Eigenvector centrality is a more sophisticated version of the degree metric, which is a simple way to measure the the influence or importance of a node [14]. The degree k_i of a node i is

$$k_i = \sum_{j=1}^{n} A_{ij}, \quad (6)$$

where A is the adjacency matrix which represents the ties between nodes i and j.

[7] https://developer.foursquare.com/docs/venues/herenow
[8] https://developer.foursquare.com/docs/responses/checkin

Since degree centrality gives a simple count of the number of ties a node has, eigenvector centrality acknowledges that not all connections are equal. Therefore, and because some edges represent stronger connections than others, the edges can be weighted. To sum up, connections to nodes which are themselves influential to others will lend a node more influence than connections to less influential nodes. Denoting the centrality of a node i by x_i, then it is possible to make x_i proportional to the average of the centralities of i's network neighbours:

$$x_i = \frac{1}{\lambda} \sum_{j=1}^{n} A_{ij} x_j, \tag{7}$$

where λ is a constant. This equation can be also rewritten defining the vector of centralities $x = (x_1, x_2, ...)$:

$$\lambda \mathbf{x} = \mathbf{A} \cdot \mathbf{x}, \tag{8}$$

where **x** is an eigenvector of the adjacency matrix with eigenvalue λ.

All these calculations are computed in the user's device. The JAMA [1] library has helped us to easily obtain the most important nodes of the graph (we have also used an online tool[9] to rapidly check the JAMA obtained values). The following method uses this library to obtain the eigenvectors to the given adjacency matrix. First the corresponding eigenvalues are estimated. Then we extract those values of the eigenvectors which are related with the highest value of the obtained eigenvalues, which corresponds to the most important node of the grid. Applying this eigenvector calculation to the A'' matrix from Section 3.1 we obtain that the highest eigenvalue $\lambda_1 = 12.502$, and the corresponding eigenvector to this eigenvalue is

$$e_1 = \begin{pmatrix} 0.569 \\ 0.491 \\ 0.401 \\ 0.392 \\ 0.349 \end{pmatrix}, \tag{9}$$

where the first value corresponds to the current user (and it is also the highest value), so we have to ignore it. The next highest value is the one we will take into account as the most important node of the grid.

We have encapsulated the Foursquare user object into a new "CompactUser" which also has a set of recommendations assigned to it, each one composed by a series of items. Taking as example the generic categories of Amazon.com we have tested our solution using a few controlled users who are friends in Foursquare and some random generated users in order to have a controlled scenario. Results are detailed in Section 4. Once the recommendations of the most important users are obtained, we upload them to a web server by Google AppEngine[10] using a simple Python service. For each user we store all possible recommendations (we manage nine main categories) and we update the estimate and the probability of fitting with his tastes. To evaluate this the developed application asks first about user's tastes among the cited categories. This information is stored in a SQLite database (this is just to evaluate the solution).

The service responses a JSON object with the recommendations and their likelihood probabilities for the user. This JSON object is parsed in the device side in order to generate the corresponding recommendations to the user.

4 RESULTS

Our solution has been evaluated by presenting to our test users the default categories that Amazon.com uses and another list with our categories recommendations. Once our users have compared both lists, they have fulfilled a questionnaire to capture their satisfaction level with the obtained results. Amazon.com default recommendations are the following:

- Kindle related products
- Clothing trends
- Products being seen by other customers
- Best watches prices
- Laptops best prices
- Top seller books

These recommendations (not categories) are not based in any user preference. On the contrary our list of recommendations establish a new order within Amazon.com's categories, indicating the probability of each one to be in the tastes of users. Navigating to Amazon.com website with privacy mode enabled allows us to see default recommendations, without taking into account any previous purchase or search. Our objective is focused in recommend items from Amazon.com categories (listed in Table 3).

Testing our solution with real users we obtained a certain approximation to their tastes. Despite the few users, check-ins and data we have access to, the system is capable of generating first generic Amazon.com recommendations (as we have already detailed in this section). Table 3 show the probabilities obtained for a user with the following tastes (see the Amazon.com whole categories listed in Table 2):

- Automotive & industrial
- Movies, music, games
- Electronics & computers
- Sports & outdoors

The middle column shows the system generated probability for each category with just an input of 3 user check-in. On the contrary the right column values corresponds to the same user doing 5 check-ins. These values are more refined, being more in accordance with the user known tastes.

Finally users have to fulfil a questionnaire rating the presented categories. This rating includes mandatory and controlled answers, from 1 to 4. This way we can compare the results with the obtained probabilities. Table 2 shows the probabilities calculated for a user from the resulting questionnaire answers. Then Table 4 compares both probabilities and calculates the approximation of each estimation.

It is important to emphasize that a new matrix is built with every user check-in. This means that previous matrices are overwritten. However, the probabilities are dynamic, and they are refined every time the user checks in a venue.

The more approximate to 0.0 ϵ is, the more accurate our solution becomes. There are some values of ϵ which shows that there are needed more check-ins to refine the obtained probabilities. On the one hand, in case of 3 check-in results the worst ones are for ”Automotive & industrial”, ”Grocery, health & beauty”, “Toys, kids & baby” and ”Sports & outdoors”. This means that if the user is interested in “Sports & outdoors” the system could not recommend any item from this category, or even worse, recommend items from ”Grocery, health & beauty”. On the other hand, 5 check-in error column is more accurate and its values are closer to 0.0. This test shows how more check-ins come out onto more refined recommendations.

[9] http://www.bluebit.gr/matrix-calculator/
[10] https://appengine.google.com/

Table 2: Results obtained from a user questionnaire answers. Left column corresponds to the recommendations names. Right column shows the taste probabilities obtained from the questionnaire.

Recommendation	Probability
Home, garden & tools (1)	0.04761904
Clothing, shoes & jewelry (2)	0.09523809
Books (3)	0.14285714
Electronics & computers (4)	0.19047619
Automotive & industrial (5)	0.14285714
Movies, music, games (6)	0.19047619
Grocery, health & beauty (7)	0.04761904
Toys, kids & baby (8)	0.04761904
Sports & outdoors (9)	0.14285714

Table 3: Results obtained from our system for one user performing 3 and 5 check-ins.

Rec.	Probability (3 check-ins)	(5 check-ins)
(1)	0.04347826	0.02222222
(2)	0.04761905	0.13333334
(3)	0.1904762	0.13333334
(4)	0.1904762	0.17391305
(5)	0.0	0.16521739
(6)	0.15	0.16382979
(7)	0.13043478	0.08510638
(8)	0.17391305	0,05319149
(9)	0.0	0,05319149

Table 4: Comparison of probabilities obtained from the questionnaire and the probabilities calculated with the proposed solution using 3 and 5 check-in results.

Rec.	App. 3 check-ins	5 check-ins	3 check-in ε	5 check-in ε
(1)	0.91304360	0.466666695	0.086956393	0.533333305
(2)	0.50000005	1.400000147	0.499999947	-0.400000147
(3)	1.33333342	0.933333399	-0.333333426	0.066666601
(4)	1.00000005	0.913043515	-0.000000053	0.086956485
(5)	0.0	1.156521753	1.0	-0.156521753
(6)	0.78750000	0.8601064	0.212499998	0.1398936
(7)	2.73913081	1.787234266	-1.73913081	-0.787234266
(8)	3.65217463	1.117021469	-2.65217463	-0.117021469
(9)	0.0	0.372340437	1.0	0.627659563

Approximation (App.) = (Solution probability / Questionnaire probability)
Deviation error (ε) = 1 - approximation

5 CONCLUSIONS AND FUTURE WORK

This paper explores the possibility of using relevant data from users' social network to alleviate the cold-user problems in a recommender system domain. The proposed solution extracts the most valuable node in the graph generated by check in a venue with an Android application using the Foursquare API. By obtaining the recommendations to this node we estimate the probability of some categories to be similar to users tastes.

In the near future we will take into account data not only from Foursquare. Other social networks with accessible APIs will be useful enough to determinate more accurately the preferred items for a user. Moreover, combining different social network analysis metrics can come out onto more accurate results. Another important aspect is to take into account more than the most valuable node for doing recommendations.

It will be also interesting to store the obtained matrices for each venue and update them with every check-in. By now matrices are not stored, so they are not dynamic, which means that a new matrix is built every time the user checks in a venue, overwriting any other previous matrix.

Finally, it becomes necessary to test the solution among a higher number of users, increasing their tastes possibilities and the offered items. We are limited by our environment because of the small amount of users and check-ins available. Therefore a dissemination of this work would be very useful to get more real data.

REFERENCES

[1] Jama : A java matrix package. `http://math.nist.gov/javanumerics/jama/`.

[2] Movielens movies recommendation service. `http://movielens.umn.edu`.

[3] Using foursquare data to redefine a neighborhood. `http://www.technologyreview.com/web/40224/`.

[4] G. Adomavicius and A. Tuzhilin, 'Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *Knowledge and Data Engineering, IEEE Transactions on*, **17**(6), 734–749, (2005).

[5] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly, 'Video suggestion and discovery for youtube: taking random walks through the view graph', in *Proceedings of the 17th international conference on World Wide Web*, pp. 895–904. ACM, (2008).

[6] P.L. Chen, C.T. Tsai, Y.N. Chen, K.C. Chou, C.L. Li, C.H. Tsai, K.W. Wu, Y.C. Chou, C.Y. Li, W.S. Lin, et al., 'A linear ensemble of individual and blended models for music rating prediction', in *KDDCup 2011 Workshop*, (2011).

[7] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin, 'Combining content-based and collaborative filters in an online newspaper', in *Proceedings of ACM SIGIR Workshop on Recommender Systems*, pp. 1–11. Citeseer, (1999).

[8] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, et al., 'The youtube video recommendation system', in *Proceedings of the fourth ACM conference on Recommender systems*, pp. 293–296. ACM, (2010).

[9] H. Kautz, B. Selman, and M. Shah, 'Referral web: combining social networks and collaborative filtering', *Communications of the ACM*, **40**(3), 63–65, (1997).

[10] I. Konstas, V. Stathopoulos, and J.M. Jose, 'On social networks and collaborative recommendation', in *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 195–202. ACM, (2009).

[11] A.N. Langville, C.D. Meyer, and P. FernÁndez, 'Google's pagerank and beyond: The science of search engine rankings', *The Mathematical Intelligencer*, **30**(1), 68–69, (2008).

[12] G. Linden, B. Smith, and J. York, 'Amazon. com recommendations: Item-to-item collaborative filtering', *Internet Computing, IEEE*, **7**(1), 76–80, (2003).

[13] P. Melville, R.J. Mooney, and R. Nagarajan, ‘Content-boosted collaborative filtering for improved recommendations’, in *Proceedings of the National Conference on Artificial Intelligence*, pp. 187–192. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, (2002).

[14] M.E.J. Newman, ‘The mathematics of networks’, *The New Palgrave Encyclopedia of Economics*, **2**, (2008).

[15] A.T. Nguyen, N. Denos, and C. Berrut, ‘Improving new user recommendations with rule-based induction on cold user data’, in *Proceedings of the 2007 ACM conference on Recommender systems*, p. 121–128, (2007).

[16] P. Resnick and H. R. Varian, ‘Recommender systems’, *Communications of the ACM*, **40**(3), 56–58, (1997).

[17] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl, ‘Item-based collaborative filtering recommendation algorithms’, in *Proceedings of the 10th international conference on World Wide Web*, p. 285–295, (2001).

[18] A.I. Schein, A. Popescul, L.H. Ungar, and D.M. Pennock, ‘Methods and metrics for cold-start recommendations’, in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, p. 253–260, (2002).

Preprocessing Algorithm for Handling Non-Monotone Attributes in the UTA method

Alan Eckhardt[1] and **Tomáš Kliegr**[2]

Abstract. UTA is a method for learning user preferences originally developed for multi-criteria decision making. UTA expects that the input attributes are monotone with respect to preferences, which limits the applicability of the method and requires manual input for each attribute. In this paper, we propose a heuristic attribute preprocessing algorithm that transforms arbitrary input attributes into a space approximately monotone with respect to user preferences, thus making it suitable for UTA. In an experimental evaluation on several real-world datasets, preprocessing the input data with the proposed algorithm consistently improved the results in comparison with the UTA-ADJ variation of the UTA STAR algorithm.

1 Introduction

The UTA (UTilités Additives) method (Jacquet-Lagreze [9]) learns an additive piece-wise linear utility model. Although a relatively old approach, it is used as a basis for many recent utility-based preference learning algorithms, e.g. [8]. UTA takes a set of alternatives ordered according to user's preferences, and learns utility functions for each attribute. Using these functions, the utility for individual attribute values are combined into the overall utility for a given object.

UTA expects that the input attributes are monotone with respect to preferences, which not only requires manual input for each attribute, but also limits the applicability of the method. In this paper, we propose a heuristic attribute preprocessing algorithm that transforms arbitrary input attributes into a space approximately monotone with respect to user preferences, thus making it suitable for UTA.

This paper is divided into four sections. Section 2 describes the UTA method with focus on its monotonicity constraints. The proposed algorithm for learning local preferences is introduced in Section 3. Experimental evaluation of the algorithm is presented in Section 4, and the concluding remarks are in Section 5.

2 Related Work

UTA method [9] draws inspiration from the way humans handle decision making tasks at a level of abstraction provided by the microeconomic utility theory. The principal inductive bias used in the UTA method is the monotonicity of utility functions. The UTA method also incorporates additional assumptions, particularly piece-wise linearity and additivity of utility functions.

UTA method aims at inferring one or more additive value functions from a given ranking (weak ordering) on the reference set X of objects[3]. Each object is described by N criteria $G = \{g_1, \ldots, g_N\}$. The evaluation scales of each criterion is given by the real-valued function $g_i : X \rightarrow [g_{i*}, g_i^*]$. The value g_{i*} is considered the worst level of the worth of the object from the decision maker's point of view with respect to criterion g_i and g_i^* the best level, $g_i(\mathbf{x})$ denotes the evaluation of object $\mathbf{x} \in X$ in criterion g_i. The method uses linear programming to find such N partial value functions u_i that best explain given preferences. The overall preference rating for an object $\mathbf{x}$ is computed as a sum of utility values across all criteria:

$$U(\mathbf{x}) = \sum_{i=1}^{N} u_i(\mathbf{x})$$

where $\mathbf{x}$ is an object and u_i are nondecreasing marginal value functions, which we call *partial utility functions*. It should be noted that the partial utility functions are piece-wise linear. For each criterion, the interval $[g_{i*}, g_i^*]$ is cut into $\alpha_i - 1$ equal intervals $[g_{i*}, g_i^2], \ldots, [g_i^{\alpha_i - 1}, g_i^*]$. This discretization is not significant for nominal criteria; for these, the number of breakpoints α_i (including the end points g_{i*}, g_i^*) corresponds to the number of values of the criteria. However, for cardinal criteria, the marginal value of an object $g_i(\mathbf{x}) \in [g_i^j, g_i^{j+1}]$ is approximated using linear interpolation between the two nearest breakpoints g_i^j, g_i^{j+1}.

The *condition of the monotonicity* requires that if for $\mathbf{x}, \mathbf{x}' \in X$, object $\mathbf{x}$ is weakly preferred to object $\mathbf{x}'$, then for another object $\mathbf{x}'' \in X$, such that $g_h(\mathbf{x}'') \geq g_h(\mathbf{x})$, for all $h = 1, \ldots, N$, object $\mathbf{x}''$ should be also weakly preferred to object $\mathbf{x}'$ [2].

In the area of Multi-Criteria Decision Making (MCDM), it is the role of the expert to derive the set of criteria from the properties of the object. The expert not only checks that the criteria meet the monotonicity requirement, but also orders the domain so that value g_{i*} is considered the worst and value g_i^* the best for criterion i.

If UTA is to be applied in wider machine learning context, rather than in the narrow area of decision support systems, a manual approach to transforming attributes to criteria is not feasible.

Kliegr in [10] proposed a non-monotonic extension of the UTA Star algorithm, called UTA-NM, which allows u_i to change direction from ascending to descending, thus allowing to use criteria[4] that do not meet the monotonicity requirement. Every change of the direction within one partial utility function is penalized to ensure that the resulting model is not overly complex and overfitted to the data.

In this paper, we investigate another option for dealing with non-monotonicity, which is based on an automatic transformation of the original, potentially non-monotonic attributes of the input objects

[1] Department of Software Engineering, Charles University in Prague, Czech Republic, email: eckhardt@ksi.mff.cuni.cz

[2] Department of Information and Knowledge Engineering, University of Economics in Prague, Czech Republic, and Multimedia and Vision Group, Queen Mary University, London, UK. email: tomas.kliegr@vse.cz

[3] Usually referred to as "actions" or "alternatives" in the UTA literature.

[4] Since the notion of criterion is closely tied with the monotonicity requirement, it would be more precise to speak of "attributes".

into criteria. The algorithm is heuristic, i.e. the resulting criteria are not guaranteed to meet the condition of monotonicity. The performance of this preprocessing step is evaluated against a baseline obtained with a non-monotonic UTA implementation coming out of UTA-NM.

3 Heuristic Algorithm for Transformation of Attributes to Criteria

Modern preference learning research deals directly with properties of input objects, an object $\mathbf{x}$ is typically described by a vector of attribute values $(x_1, \ldots, x_N)$, no special requirements on the attribute values are typically made [7]. In contrast, the original UTA model, and MCDA in general, abstracts from the original attributes of the object. It is already assumed that the *expert* has used these attributes to construct the criteria set G; each object is represented by criteria values $(g_1(\mathbf{x}), \ldots, g_N(\mathbf{x}))$. The individual criteria are assumed to meet several properties, including the condition of monotonicity introduced in the previous section.[5]

In this section, we present a heuristic algorithm that transforms the original values of the attributes, so that the result better meets the key requirement put on *criteria* addressed in this paper – the condition of monotonicity. Since the input for the algorithm is the preference rating of one user[6], we call it *local preference* transformation.

Using D_{A_i} to denote the domain of the original attribute i, the local preferences transformation can be viewed as function $f_i : D_{A_i} \rightarrow [g_{i*}, g_i^*]$, which is a "concretization" of the criterion function $g_i : X \rightarrow [g_{i*}, g_i^*]$ in the formal UTA model.

Each object $\mathbf{x}$ originally described by attribute values $x_1, \ldots, x_N$ is now described by values $f_1(x_1), \ldots, f_N(x_N)$. The intuition is that $f_i(x_i)$ is an "average" of ratings of the particular user across objects that have value x_i in attribute i, we denote this average as $r(\mathbf{x})$. Since the preference information which is consumed by the UTA method is in the form of weak order of alternatives rather than ratings, for the purpose of the local transformation algorithm this weak order needs to be transformed to ratings. A straightforward approach is to assign values in the $[0, 1]$ range corresponding to the position of the object in the weak preference order, with the best ranked object retrieving the highest rating 1 and the worst ranked object the lowest ranking 0, with the remaining ranks equidistantly distributed in the $[0, 1]$ range.

The definition of the transformation function f_i depends on the type of the input attribute, which can be either nominal or cardinal. In our present work ordinal attributes are not addressed, however, they can be cast to integer and handled as a cardinal data type. In the remainder of this section, we discuss definition of the local transformation function f_i first for nominal and then for cardinal attributes.

3.1 Nominal Attributes

"Representants" is a method for finding preferences over nominal attributes, which we proposed in [4]. It is based on the analysis of the distribution of ratings. The local preference function for a nominal attribute A_i has the following form:

$$f_i(a) = \frac{\sum_{\{\mathbf{x}|x_i=a\}} r(\mathbf{x})}{|\{\mathbf{x}|x_i = a\}|},$$

where a is an attribute value.

Figure 1 illustrates the computation on the notebook choice problem. Ratings of three hypothetical users with respect to the nominal *notebook colour* attribute are presented. For each user and colour, a frequency distribution of ratings is shown. Let us focus on one value only, e.g. black. Black-coloured notebooks are highly preferred for users 1 and 2. The vertical line in the histogram represents the average rating of each object with colour black - for both users, the average is close to 1. The average will be used as the *representative* rating of the value black for the given user, or in other words, preference of black colour.

Formally, the set of black objects is denoted as

$$\{\mathbf{x}|x_{colour} = black\}$$

and the sum of ratings of black objects is then

$$\sum_{\{\mathbf{x}|x_{colour}=black\}} r(\mathbf{x})$$

.

The resulting representative ratings do not necessarily maintain the monotonicity of ratings. Consider five objects (given in the order of increasing stated preference by the user) $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5$ with two attributes *colour* and *price*. The description of these objects with respect to these attributes is given by Table 1. Once the stated preference, originally given in terms of stars, is converted to a numerical rating, the original attributes can be transformed with the local preferences function obtaining $f_{colour}(black) = 0.375$ and $f_{colour}(red) = 0.58$, $f_{cpu}(intel) = 0.25$ and $f_{cpu}(amd) = 0.5$, $f_{cpu}(motorola) = 1.0$.

This example can be also used to illustrate the fact that the result of the local preferences transformation is not guaranteed to not violate the condition of monotonicity: $\mathbf{x}_4$ is preferred to $\mathbf{x}_3$, for $\mathbf{x}_2$ it holds that $g_h(\mathbf{x}_2) \geq g_h(\mathbf{x}_4)$, for all $h = 1, \ldots, N$, therefore $\mathbf{x}_2$ should also be weakly preferred to $\mathbf{x}_3$. Since this is not satisfied, the condition of monotonicity is violated.

Table 1. Notebook choice example – nominal attributes.

	object				
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$
stated order	*	**	***	****	*****
rating	0	0.25	0.5	0.75	1
	original attribute values				
colour	black	red	red	black	red
cpu	intel	amd	intel	amd	motorola
	transformed values (criteria)				
colour	0.375	0.58	0.58	0.375	0.58
cpu	0.25	0.5	0.25	0.5	1

3.2 Cardinal Attributes

This section describes the proposed approach for transformation of cardinal (numerical) attributes.

Linear regression is a very useful method that finds a relation in a data set in the form of a linear function. In the local preferences transformation, *univariate* linear regression is used to find a relationship between each cardinal input attribute and the rating as the dependent variable. Expanding our notebook example, consider additional *notebook price* attribute. Applying the local transformation for cardinal attributes, notebook price is used as a regressor and the

[5] In this paper, we make the assumption that the original N input attributes are converted to the same number of criteria.

[6] Decision maker in the MCDA terminology.

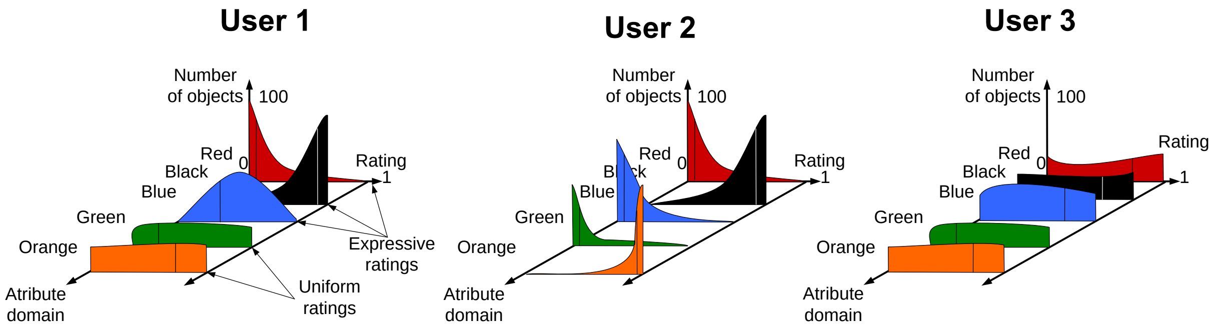


Figure 1. Illustrative ratings of three users for the notebook colour attribute.

user *rating* as the dependent variable. The result of the regression is a linear function of the form:

$$f_{price}(price) = \alpha * price + \beta$$

.

If the underlying attribute is of so called *cost* type, i.e. user preference decreases with increasing attribute value, a direct use of the attribute value x_i in place of the criterion value $g_i(\mathbf{x})$ would violate the condition that the value g_{i*} is considered the worst level of the worth of the object from the decision maker's point of view with respect to criterion g_i and g_i^* the best level (Figure 2b). Linear regression will generally solve this problem, rearranging the values appropriately.

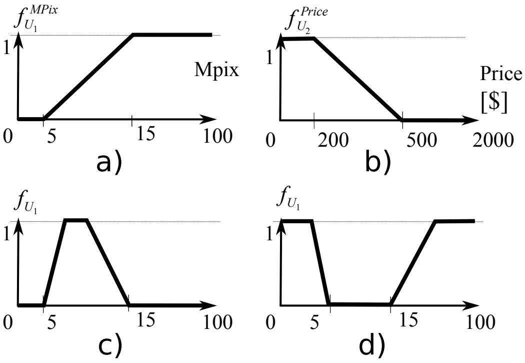


Figure 2. Four basic types of preference over cardinal domains

However, the weakness of linear regression is that it finds only linear orderings, therefore it does not generally fix a violation of the condition monotonicity. Figure 2 shows four typical types of preferences over cardinal attributes. Only two of them are linear. Linear regression is not able to find "hill" and "valley" types. These types can be learned using our method "Peak" proposed in [5].

We have considered also other possibilities for finding preferences over cardinal attributes. In our preliminary experiments on small training sets, using quadratic regression consistently worsened the results.

4 Experiments

4.1 Performance measures

We will use the following notation: $r(o)$ is the user's rating of object o, $\widehat{r}(o)$ is the rating predicted by the method.

Tau coefficient expresses the similarity of two ordered lists L_1, L_2. In our case, the first list L_1 is ordered according to the user's rating and the second list L_2 according to the rating estimated by the method. The lists are sorted in decreasing order, so that the most preferred objects are on the top of the lists. In the simplest case, the lists consist only of ids of objects. The tau coefficient is then computed according to the number of concordant pairs. A pair of objects o,p is concordant, if either o is before p in both lists, or p is before o in both lists. A pair that is not concordant, is discordant. Then, tau coefficient can be computed as follows:

$$\tau(L_1, L_2) = \frac{n_c - n_d}{1/2 * n * (n-1)},$$

where n_c is the number of concordant pairs and n_d is the number of discordant pairs. δ in following formula stands for Kronecker delta - $\delta(condition) = 1$ if *condition* is true, 0 otherwise.

$$n_c = \sum_{o,p \in \mathcal{X}} \delta(sgn(L_1(o) - L_1(p)) = sgn(L_2(o) - L_2(p)))$$

Another measure expressing the similarity of two lists is Pearson correlation. This measure is often used in machine learning for studying the dependency of two variables. If the correlation is high, the two lists are similar, if the correlation is low, the lists are different. The value of the correlation coefficient ranges from -1 to 1. Value -1 means lists with exactly inverse ordering, 1 corresponds to the same ordering. Correlation 0 means there is no connection between the two orderings.

$$corr(r, \widehat{r}) = \frac{\sum\limits_{o \in \mathcal{X}} (r(o) - \bar{r})(\widehat{r}(o) - \bar{\widehat{r}})}{(n-1)s_r s_{\widehat{r}}},$$

where $\bar{r}$ is the average rating, $\bar{\widehat{r}}$ is the average predicted rating, s_r is the standard deviation of the original ratings and $s_{\widehat{r}}$ is the standard deviation of the predicted ratings.

UTA method is focused on preserving the order of objects rather than estimating their ratings, so it does not make sense to use the Root Mean Square Error (RMSE) metric, which is commonly used for other methods.

Two measures of computational performance will be studied. The first measure is the time required to train the method on a set of ranked objects. The second one is the time required to evaluate a testing object.

Table 2. Datasets with monotone class attribute from the UCI repository.

Dataset name
Car Evaluation
Contraceptive Method Choice
Nursery
Poker Hand
Post-Operative Patient
Teaching Assistant Evaluation
Wine
Wine Quality[1]

4.2 Datasets

UCI [6] contains a number of datasets with different properties. We were most concerned in classification tasks, where there is an ordering of classes. For evaluation of the UTA method, only datasets suitable for the object ranking task were relevant. E.g. the famous Iris dataset were excluded, because we are not able to order the three classes - Iris Setosa, Iris Versicolour and Iris Virginica - in a meaningful way. Every classification dataset of UCI was studied for the presence of monotonicity in the class attribute. The chosen datasets from UCI repository are in Table 2.

We acknowledge that using UCI for validation of user preference learning methods may not give representative results, since these are not real-world preference datasets.

4.3 Experimental Implementation

For the experimental evaluation, we used our implementation of non-monotonic UTA called UTA-ADJ. It is a based on similar ideas as the UTA-NM algorithm, introduced in our earlier work [10]. UTA-NM removes the monotonicity constraints imposed by the UTA Star algorithm. A penalization element is added to prevent overfitting by excessive number of changes in shape of partial utility functions, however, the penalization entails an excessive computational cost.

To remedy the computational issue, UTA-ADJ takes a different route to allow non-monotonicity in partial utility functions. UTA-ADJ runs the standard UTA Star algorithm multiple times, gradually testing the influence of placing a change of shape at individual breakpoints across all partial utility functions. The breakpoint in which the change of shape yields the largest increase in the objective function in comparison with the baseline, is retained.

This procedure is repeated until the preset threshold of maximum number of changes in shape is retained or the improvement in the objective function is lower than a preset minimum increase. In the first iteration, the baseline is the objective value attained by a fully monotonic run of the UTA Star algorithm, in subsequent iterations it is the best objective value from the previous iteration.

The computational costs of multiple runs of the UTA Star algorithm is in our experience much lower than the cost of a single UTA-NM run with the penalization constraints. Nevertheless, it should be noted that the computational advantage of UTA-ADJ over UTA-NM is inversely related to the maximum number of changes in shape threshold.

UTA-ADJ algorithm closely resembles the method proposed by Despotis and Zopounidis [3], with the difference that breakpoints, where the change of shape occurs, are not externally set parameters, but are determined automatically. Also, more changes of shape per partial utility functions are allowed.

The UTA-ADJ implementation is available as an interactive online application at `http://nlp.vse.cz/uta`.

4.4 Experimental Results

This section describes our experiments on the UCI datasets. The results are averaged across all the datasets listed in Table 2. For the given training set size, 20 different (randomized) training sets were used for each dataset. We compare UTA-ADJ with the (possibly) improved UTA+Local, which is also an UTA-ADJ run, but involving the local preferences transformation. The maximum number of changes in shape threshold was set to 2 for all experiments.

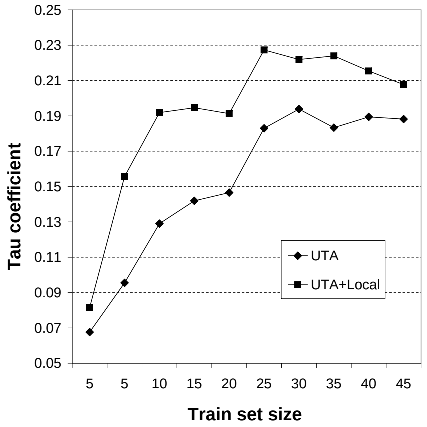


Figure 3. Tau rank coefficient for all datasets.

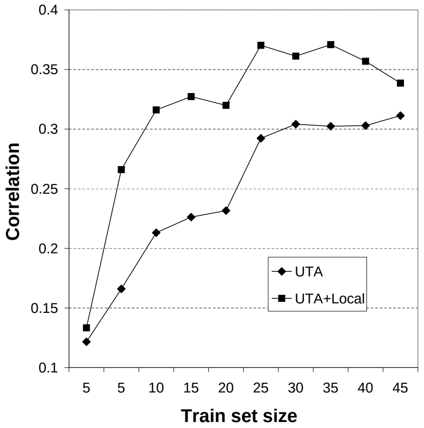


Figure 4. Correlation for all datasets.

Figure 3 presents a comparison of performance on individual datasets in terms of the tau coefficient. Here, the advantage of using local preferences is clear, the improvement of the value of tau coefficient for moderate train size is around 0.05. A comparable result for Pearson correlation can be observed in Figure 4. The relative increase in correlation and tau coefficient is about 28%.

The results are convincing - using local preferences with the UTA-ADJ variant of the UTA method significantly improved its performance in all performance measures compared to UTA-ADJ only baseline. Moreover, the time to train the classifier has also decreased. All results were significant at the level $p < 0.05$.

5 Conclusion

We proposed a preprocessing algorithm called *local preferences transformation*, which allows to use the UTA method with non-monotone attributes.

The experimental results confirm the benefits of the proposed approach as a preprocessing step for UTA-ADJ, a variant of the UTA method, which already has some adjustments to handle nonmonotone attributes. We assume the effectiveness of our local preferences preprocessing algorithm to hold also for the common UTA Star algorithm. Nevertheless, an experimental evaluation of this hypothesis is a priority for future work.

A promising direction for extending our research is using more elaborate methods than linear regression for preprocessing cardinal attributes. Regarding nominal attributes, further work should be directed at investigation of circumstances, in which the proposed algorithm is and is not effective in maintaining the condition of monotonicity.

It would also be interesting to find out the relation between the degree of monotonicity of the data and the performance of UTA with/without applying the proposed local preferences transformation in the preprocessing phase.

The UTA method implementation used in the experiments is available in a form of an interactive web application at `http://nlp.vse.cz/uta`. We plan to make the local transformation algorithm available to the community by integrating it with this software.

Acknowledgment

We would like to thank the reviewers for their comments, which helped to improve the quality of this paper considerably. This research was supported by projects SVV-2010-261312, MŠM 0021620838 and grant GAČR P202/10/0761.

REFERENCES

[1] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis, 'Modeling wine preferences by data mining from physicochemical properties', *Decision Support Systems*, (June 2009).

[2] Krzysztof Dembczyński, Wojciech Kotłowski, Roman Słowiski, and Marcin Szelag, 'Learning of rule ensembles for multiple attribute ranking problems', in *Preference Learning*, eds., Johannes Fürnkranz and Eyke Hüllermeier, 217–247, Springer-Verlag, (2010).

[3] D.K. Despotis and Constantin Zopounidis, 'Building additive utilities in the presence of non-monotonic preference', in *Advances in Multicriteria Analysis*, eds., P.M. Pardalos, Y. Siskos, and C. Zopounidis, 101–114, Kluwer Academic Publisher, Dordrecht, (1993).

[4] Alan Eckhardt and Peter Vojtáš, 'Considering data-mining techniques in user preference learning', in *2008 International Workshop on Web Information Retrieval Support Systems*, pp. 33–36, (2008).

[5] Alan Eckhardt and Peter Vojtáš, 'Learning user preferences for 2cp-regression for a recommender system', in *SOFSEM '10: Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science*, pp. 346–357, Berlin, Heidelberg, (2010). Springer-Verlag.

[6] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[7] Johannes Fürnkranz and Eyke Hüllermeier, *Preference Learning*, Springer, 2010.

[8] Salvatore Greco, Vincent Mousseau, and Roman Slowinski, 'Ordinal regression revisited: multiple criteria ranking using a set of additive value functions', *European Journal of Operational Research*, **191**(2), 415–435, (December 2008).

[9] E. Jacquet-Lagreze and Yannis Siskos, 'Assessing a set of additive utility functions for multicriteria decision-making, the uta method', *European Journal of Operational Research*, **10**(2), 151–164, (June 1982).

[10] Tomáš Kliegr, 'UTA - NM: Explaining stated preferences with additive non-monotonic utility functions', in *Proceedings of Workshop Preference Learning in ECML/PKDD'09*, (September 2009).

Learning from Pairwise Preference Data using Gaussian Mixture Model

Mihajlo Grbovic and **Nemanja Djuric** and **Slobodan Vucetic**[1]

Abstract. In this paper we propose a fast online preference learning algorithm capable of utilizing incomplete preference information. It is based on a Gaussian mixture model that learns soft pairwise label preferences via minimization of the proposed soft rank loss measure. Standard supervised learning techniques, such as gradient descent or Expectation Maximization can be used to find the unknown model parameters. Algorithm outputs are soft pairwise label preference predictions that need to be further aggregated to produce a total label ranking prediction, for which several existing algorithms can be used. The main advantages of the proposed learning algorithm are the ability to process a single training instance at a time, low time and space complexity, ease of implementation, and model reuse.

1 INTRODUCTION

Label Ranking is emerging as an important and practically relevant preference learning field. Unlike the standard problems of classification and regression, label ranking learning is a complex learning task, which involves the prediction of strict label order relations, rather than single values. Specifically, in the label ranking scenario, each instance, which is described by a set of features $\mathbf{x}$, is assigned a ranking of labels π, that is a total (e.g. $\pi = (3, 5, 1, 4, 2)$) or partial (e.g. $\pi = (5, 3, 2)$) order over a finite set of class labels $\mathcal{Y}$ (e.g. $\mathcal{Y} = \{1, 2, 3, 4, 5\}$). The label ranking problem consists of learning a model that maps instances $\mathbf{x}$ to a total label order $h : \mathbf{x}_n \rightarrow \pi_n$. It is assumed that a sample from the underlying distribution $D = \{(\mathbf{x}_n, \pi_n), n = 1, ..., N\}$, where $\mathbf{x}_n$ is a d-dimensional feature vector and π_n is a vector containing a total or partial order of a finite set $\mathcal{Y}$ of L class labels, is available for training.

This problem has recently received a lot of attention in the machine learning community and has been extensively studied [6, 4, 9, 3, 16]. A survey of recent label ranking algorithms can be found in [8].

There are many practical applications in which the objective is to learn an exact label preference of an instance in form of a total order. For example, in the case of document categorization, where it is very likely that a document belongs to multiple topics (e.g. sports, entertainment, baseball, etc.), one might not be interested only in predicting which topics are relevant for a specific document, but also to rank the topics by relevance. Additional applications include: meta-learning [18], where, given a new data set, the task is to induce a total rank of available algorithms according to their suitability based on the data set properties; predicting food preferences for new costumers based on the survey results, demographics, and other characteristics of respondents [12]; determining an order of questions in a survey for a specific user based on respondent's attributes. A recent publication [14], suggests clustering of label ranking data, which could be of great practical importance, especially in target marketing.

There are three principal approaches for label ranking. The first decomposes label ranking problem into one or several binary classification problems. Ranking by pairwise comparison (PW) [11], for example, creates $L \cdot (L - 1)/2$ classification problems, one for each possible pairwise ranking. Pairwise binary classifier predictions are aggregated into a total label order by voting. The constraint classification (CC) [9], on the other hand, transforms the label ranking problem into a single binary classification problem by augmenting the data set, such that each example $\mathbf{x}$ is mapped into $L \cdot (L - 1)/2$, $(d \times L)$-dimensional, examples. This allows for training of a single classifier.

The second approach is to use utility functions, where the goal is to learn mappings $f_k : X \rightarrow R$ for each label $k = 1, ..., L$, which assign a value $f_k(\mathbf{x})$ to each label, such that $f_i(\mathbf{x}) < f_j(\mathbf{x})$ if $\mathbf{x}$ prefers label j over i. For example, the label ranking method proposed in [6] represents each f_k as a linear combination of base ranking functions. The utility functions are learned to minimize the number of ranking errors and the final rank is produced simply by ranking the utility scores. It should be noted that the utility function-based approach is also popular in the related object ranking problem, where techniques based on SVM [10] and AdaBoost [7] have been proposed.

The third approach is represented by a collection of algorithms which use probabilistic approaches for label ranking, such as the ones that rely on the Mallows [13] and the Plackett-Luce (PL) [15] models. A typical representative is the instance-based (IB) label ranking [4, 3]. Given a new instance $\mathbf{x}$, the k-Nearest Neighbor algorithm is used to locate its neighbors in the feature space. Then, the neighbors' label rankings are aggregated to provide prediction. Rank aggregation for prediction is not a trivial task, particularly in presence of partial label ranks. Mallows [4] and Plackett-Luce [3] models that describe probability distribution of rankings have been used to come up with the optimization criterion for rank aggregation. As an alternative, CPS probabilistic model for rank aggregation has been recently proposed [16].

Instance-based label ranking algorithms are simple and intuitive. Furthermore, they have been shown to outperform the competitors in various label ranking scenarios. However, their success comes at a large cost associated with both memory and time. First, they require that the entire training data set is stored in memory, which can be costly or even impossible in the resource-constrained applications. Storing the original data can also raise privacy issues as the data might contain sensitive user information. Second, the prediction involves costly nearest neighbor search and aggregation of neighbors' label rankings. The aggregation is slow as it requires using op-

[1] Temple University, Department of Computer and Information Sciences, Philadelphia, USA, email: {mihajlo.grbovic, nemanja.djuric, slobodan.vucetic }@temple.edu

timization techniques at prediction time, such as iterative Minorization Maximization(IB-PL) or exhaustive search (IB Mallows).

In this paper we propose an online, time- and memory-efficient algorithm for learning label preferences based on the Gaussian Mixture Model (GMM), which could be attractive because of an intuitively clear learning process and ease of implementation. The model preserves privacy as it consists of mixtures defined by prototypes which are not the actual data points. Every prototype is associated with preference judgments for each pair of labels. Unlike many competitors, our algorithm is not limited to a specific type of label ranking and could support various ranking structures (bipartite, multipartite, etc). For an unlabeled instance, GMM predicts the soft label preferences by averaging prototypes' pairwise preferences according to distances.

These soft label preferences in form of a preference matrix need to be aggregated further, into a total order of labels. This is a well known problem in preference learning and is an especially popular research area in the object ranking scenario, where numerous methods have been proposed [5, 1, 2].

2 PRELIMINARIES

In the label ranking scenario, a single instance, described by a d-dimensional vector $\mathbf{x}$, is associated with a total order of assigned class labels, represented as a permutation π of the set $\{1, ..., L\}$, where L is the number of available class labels. We define π such that $\pi(i)$ is the class label at i-th position in the order, and $\pi^{-1}(j)$ is a position of the y_j class label in the order. The permutation can also describe incomplete ranking $\{\pi(1), ..., \pi(k)\} \subset \{1, ..., L\}, k < L$.

In our approach, instead of the total order, we use a zero-diagonal preference matrix $\mathbf{Y}$. When a preference between labels y_i and y_j exists, we set $\mathbf{Y}(i,j) > \mathbf{Y}(j,i)$ if y_i is preferred over y_j and $\mathbf{Y}(i,j) < \mathbf{Y}(j,i)$ otherwise, $\mathbf{Y}(i,j) + \mathbf{Y}(j,i) = 1$, for $i, j \in \{1, ..., L\}$. A value of $\mathbf{Y}(i,j)$ which is close to 1 is interpreted as a strong preference that y_i should be ranked above y_j. A typical approach is to assign $\mathbf{Y}(i,j) = 1$ and $\mathbf{Y}(j,i) = 0$ if $y_i \succ_{\mathbf{x}} y_j$. Similarly, uncertain (soft) preferences can be modeled by using values lower than 1. For example, indifferences (ties) are represented by setting $\mathbf{Y}(i,j) = \mathbf{Y}(j,i) = 0.5$. In case of non-existing, incomparable or missing preferences, both $\mathbf{Y}(i,j) = 0$ and $\mathbf{Y}(j,i) = 0$.

This representation allows us to work with complete and partial label orders, as well as with pairwise preferences with uncertainties and indifferences. Finally, bipartite and multi-partite label rankings could be handled as well.

Evaluation metrics. Let us assume that N historical observations are collected in a form of a data set $D = \{(\mathbf{x}_n, \mathbf{Y}_n), n = 1, ..., N\}$. The objective in all scenarios is to train a ranking function $h : \mathbf{x}_n \to \hat{\pi}_n$ from data set D that outputs a total label order.

In the Label Ranking scenario, to measure the degree of correspondence between true and predicted rankings for n-th example, π_n and $\hat{\pi}_n$ respectively, it is common to use the Kendall's tau distance $d_n =| \{(y_i, y_j) : \pi_n^{-1}(y_i) > \pi_n^{-1}(y_j) \wedge \hat{\pi}_n^{-1}(y_j) > \hat{\pi}_n^{-1}(y_i)\} |$. To evaluate a label ranking model, the label ranking loss on the data set D is defined as the average normalized Kendall's tau distance,

$$loss_{LR} = \frac{1}{N} \sum_{n=1}^{N} \frac{2 \cdot d_n}{L \cdot (L-1)}. \qquad (1)$$

Note that the measure simply counts the number of discordant label pairs and reports the average over all considered pairwise rankings. Given the general preference matrix representation, assuming binary matrix predictions $\hat{\mathbf{Y}}_n$, we can rewrite (1) as

$$loss_P = \frac{1}{N} \sum_{n=1}^{N} \frac{\| \mathbf{Y}_n - \hat{\mathbf{Y}}_n \|_F^2}{L \cdot (L-1)}, \qquad (2)$$

where $\| \cdot \|_F$ is Frobenius matrix norm. Indeed, for each example n, the square of the Frobenius norm sums up to double the number of discordant label pairs.

For models with soft label preference predictions $\hat{\mathbf{Y}}_n$, e.g., $\hat{\mathbf{Y}}_n(i,j) = 0.7, \hat{\mathbf{Y}}_n(j,i) = 0.3$, loss (2) can be interpreted as a soft version of (1).

We can solve the preference learning task in two stages. In the learning stage, function $f : \mathbf{x}_n \to \mathbf{Y}_n$ is learned via minimizing (2). In the aggregation stage, given the model predictions in a form of $\mathbf{Y}_n$, the total order prediction π_n is computed using a preference aggregation mapping $g : \mathbf{Y}_n \to \pi_n$. In the next section we show the details of the proposed Gaussian Mixture Model algorithm to be used in the learning stage. Existing algorithms such as [5, 1, 2], can be used in the aggregation stage.

3 GAUSSIAN MIXTURE MODEL FOR LABEL RANKING

The GMM model for label ranking is completely defined by a set of K mixtures, i.e., prototypes $\{(\mathbf{m}_k, \mathbf{Q}_k), k = 1, ..., K\}$, where $\mathbf{m}_k$ is a d-dimensional vector in input space and $\mathbf{Q}_k$ is the corresponding preference matrix.

First, we introduce the probability $\mathbb{P}(k \mid \mathbf{x})$ of assigning observation $\mathbf{x}$ to k-th prototype that is dependent on their (Euclidean) distance. Let us assume that the probability density $\mathbb{P}(\mathbf{x})$ of $\mathbf{x}$ can be described by a mixture model,

$$\mathbb{P}(\mathbf{x}) = \sum_{k=1}^{K} \mathbb{P}(\mathbf{x} \mid k) \cdot \mathbb{P}(k), \qquad (3)$$

where K is the number of prototypes, $\mathbb{P}(k)$ is the prior probability that a data point is generated by k-th prototype, and $\mathbb{P}(\mathbf{x} \mid k)$ is the conditional probability that k-th prototype generates particular data point $\mathbf{x}$. Let us represent the conditional density function $\mathbb{P}(\mathbf{x} \mid k)$ with the normalized exponential form $\mathbb{P}(\mathbf{x} \mid k) = \theta(k) \cdot \exp(f(\mathbf{x}, \mathbf{m}_k))$ and consider a Gaussian mixture with $\theta(k) = (2\pi\sigma_p^2)^{-1/2}$ and $f(\mathbf{x}, \mathbf{m}_k) = -\|\mathbf{x} - \mathbf{m}_k\|^2 / 2\sigma_p^2$. We assume that all prototypes have the same standard deviation σ_p and the same prior, $\mathbb{P}(k) = 1/K$. Given this, using the Bayes' rule we can write the assignment probability as

$$\mathbb{P}(k \mid \mathbf{x}) = \frac{\exp(-\|\mathbf{x} - \mathbf{m}_k\|^2 / 2\sigma_p^2)}{\sum_{u=1}^{K} \exp(-\|\mathbf{x} - \mathbf{m}_u\|^2 / 2\sigma_p^2)}. \qquad (4)$$

To derive a cost function, we propose the following mixture model for the posterior probability $\mathbb{P}(\mathbf{Y} \mid \mathbf{x})$,

$$\mathbb{P}(\mathbf{Y} \mid \mathbf{x}) = \sum_{k=1}^{K} \mathbb{P}(k \mid \mathbf{x}) \cdot \mathbb{P}(\mathbf{Y} \mid k). \qquad (5)$$

Based on this model, example $\mathbf{x}$ is assigned to the prototypes probabilistically and its preference matrix is a weighted average of the prototype preference matrices. The mixture model assumes the conditional independence between $\mathbf{x}$ and $\mathbf{Y}$ given k, $\mathbb{P}(\mathbf{Y} \mid \mathbf{x}, k) = \mathbb{P}(\mathbf{Y} \mid k)$. For $\mathbb{P}(k \mid \mathbf{x})$ we assume the Gaussian distribution from

(4). For the probability of generating a preference matrix $\mathbf{Y}$ by prototype k, $\mathbb{P}(\mathbf{Y} \mid k)$, we also assume Gaussian error model with mean $(\mathbf{Y} - \mathbf{Q}_k)$ and standard deviation σ_y. The resulting cost function $l(\lambda)$ can be written as the negative log-likelihood,

$$l(\lambda) = -\frac{1}{N}\sum_{n=1}^{N} \ln \sum_{k=1}^{K} \mathbb{P}(k \mid \mathbf{x}) \cdot \mathcal{N}(\mathbf{Y} - \mathbf{Q}_k, \sigma_y^2), \qquad (6)$$

where $\lambda = \{\mathbf{m}_k, \mathbf{Q}_k, k = 1, ..., P, \sigma_p, \sigma_y\}$ are the model parameters. For the compactness of notation, let us define $g_{nk} = \mathbb{P}(k \mid \mathbf{x}_n)$ and $e_{nk} = \mathcal{N}(\mathbf{Y}_n - \mathbf{Q}_k, \sigma_y^2)$.

It is important to observe that, after proper normalization, (6) reduces to (2) if examples are assigned to prototypes deterministically. Therefore, it can be interpreted as its soft version. If prototype matrices $\mathbf{Q}_k$ consisted of only 0 and 1 entries (hard label preferences) (6) further reduces to (1).

The objective is to estimate the unknown model parameters, namely prototype positions $\mathbf{m}_k$ and their preference matrices $\mathbf{Q}_k, k = 1, ..., K$. This is done by minimizing the cost function $l(\lambda)$ with respect to the parameters. This can be achieved in several different ways. If online learning capability is a requirement, one can use the stochastic gradient descent method and obtain the learning rules by calculating derivatives $\partial l(\lambda)/\partial \mathbf{m}_k$ and $\partial l(\lambda)/\partial \mathbf{Q}_k$ for $k = 1, ..., K$. This results in following rules for n-th training example,

$$\begin{aligned} \mathbf{m}_k^{n+1} &= \mathbf{m}_k^n - \alpha(n)\frac{(L_n - e_{nk}) \cdot g_{nk}}{L_n}\frac{(\mathbf{x}_n - \mathbf{m}_k)}{\sigma_p^2} \\ \mathbf{Q}_k^{n+1} &= \mathbf{Q}_k^n - \alpha(n)\frac{e_{nk} \cdot g_{nk}}{L_n}\frac{(\mathbf{Y}_n - \mathbf{Q}_k)}{\sigma_y^2}, \end{aligned} \qquad (7)$$

where $L_n = \sum_{k=1}^{P}(g_{nk} \cdot e_{nk})$ and $\alpha(n)$ is the learning rate.

The resulting model has complexity $\mathcal{O}(NKL)$. Otherwise, the problem can straightforwardly be mapped into Expectation-Maximization (EM) framework following the procedure from [17].

Initialization is done by selecting the first P training points as the initial prototypes. If any $\mathbf{Q}_k$ prototype preference matrix obtained in such manner contains empty elements, they are replaced with 0.5 entries, as the corresponding labels will initially be treated equally.

GMM model generalizes the training data to produce a representation in terms of prototype vectors and effectively utilizes distances to prototypes as a similarity measure to calculate the predicted label rank. When compared to IB-based algorithms, GMM is a more global model, that aggregates over more data, thus also alleviating influence of noise. Therefore, it is expected to outperform IB-based algorithms, whose performance is highly dependent on the quality of the training data and the presence of outliers, since no abstraction is made during the training phase. We could try to aggregate over more data by considering a large number of neighbors in IB algorithms, however, by doing so we start ignoring distances between the query instance and its neighbors as a similarity measure. This remains to be seen after a proper experimental evaluation.

A disadvantage of the GMM model is that it requires aggregation of the predicted label preference matrix to produce a total order of labels. Luckily, most of the existing algorithms have low complexity, e.g. $\mathcal{O}(L \log L)$ for QuickSort [1]. In our future work we plan to evaluate the pros and cons of different aggregation methods.

4 CONCLUSION AND FUTURE WORK

We introduced an idea of a Gaussian Mixture Model algorithm for Label Ranking. The main advantages of the new method are: (1) it is capable of operating in an online manner, (2) it is memory-efficient since it operates on a predefined budget, (3) it preserves privacy, (4) it could potentially reuse the model when new labels are introduced. There are several avenues which need to be pursued further: (1) experimental evaluation of the proposed method on benchmark data (2) determining the optimal number of prototypes K using statistical learning theory, (3) low rank approximation of pairwise preference matrices to reduce memory requirements, (4) evaluating different preference matrix aggregation algorithms, (5) applying the algorithm to clustering of label ranking data. In the future work we plan to address these issues.

REFERENCES

[1] Nir Ailon and Mehryar Mohri, 'An Efficient Reduction of Ranking to Classification', in *COLT*, pp. 87–98. Omnipress, (2008).

[2] Maria-Florina Balcan, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B. Sorkin, 'Robust Reductions from Ranking to Classification', in *COLT*, volume 4539 of *Lecture Notes in Computer Science*, pp. 604–619, (2007).

[3] Weiwei Cheng, Krzysztof Dembczyński, and Eyke Hüllermeier, 'Label ranking methods based on the Plackett-Luce model', in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 215–222, (2010).

[4] Weiwei Cheng, Jens Hühn, and Eyke Hüllermeier, 'Decision tree and instance-based learning for label ranking', in *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, pp. 161–168, (2009).

[5] William W. Cohen, Robert E. Schapire, and Yoram Singer, 'Learning to order things', *Journal of Artificial Intelligence Research*, **10**, 243–270, (1999).

[6] Ofer Dekel, Christopher Manning, and Yoram Singer, 'Log-Linear Models for Label Ranking', in *Advances in Neural Information Processing Systems*, volume 16, MIT Press, (2003).

[7] Yoav Freund, Raj D. Iyer, Robert E. Schapire, and Yoram Singer, 'An Efficient Boosting Algorithm for Combining Preferences', *Journal of Machine Learning Research*, **4**, 933–969, (2003).

[8] Thomas Gärtner and Shankar Vembu, 'Label Ranking Algorithms: A Survey', in *Preference Learning*, ed., Eyke Hüllermeier Johannes Fürnkranz, Springer–Verlag, (2010). (to appear).

[9] Sariel Har-Peled, Dan Roth, and Dav Zimak, 'Constraint classification for multiclass classification and ranking', in *Proceedings of the 16th Annual Conference on Neural Information Processing Systems, NIPS-02*, pp. 785–792. MIT Press, (2003).

[10] R. Herbrich, T. Graepel, and K. Obermayer, 'Large Margin Rank Boundaries for Ordinal Regression', in *Advances in Large Margin Classifiers*, pp. 115–132. MIT Press, (2000).

[11] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker, 'Label ranking by learning pairwise preferences', *Artificial Intelligence*, **172**, 1897–1916, (2008).

[12] Toshihiro Kamishima and Shotaro Akaho, 'Efficient Clustering for Orders', in *ICDM Workshops*, pp. 274–278. IEEE Computer Society, (2006).

[13] C. L. Mallows, 'Non-null ranking models', *Biometrika*, **44**, 114–130, (1967).

[14] Grbovic Mihajlo, Nemanja Djuric, and Slobodan Vucetic, 'Supervised clustering of label ranking data', *SIAM International Conference on Data Mining*, (2012).

[15] R. L. Plackett, 'The analysis of permutations', *Applied Statistics*, **24**(2), 193–202, (1975).

[16] Tao Qin, Xiubo Geng, and Tie-Yan Liu, 'A New Probabilistic Model for Rank Aggregation', in *Advances in Neural Information Processing Systems 23*, 1948–1956, MIT Press, (2010).

[17] Weigend A. S., Mangeas M., and Srivastava A. N., 'Nonlinear Gated Experts for Time Series: Discovering Regimes and Avoiding Overfitting', *International Journal of Neural Systems*, **6**, 373–399, (1995).

[18] Ricardo Vilalta and Youssef Drissi, 'A perspective view and survey of meta-learning', *Artificial Intelligence Review*, **18**, 7795, (2002).

Preference Function Learning over Numeric and Multi-valued Categorical Attributes

Lucas Marin[1], **Antonio Moreno** and **David Isern**

Abstract. One of the most challenging goals of recommender systems is to infer the preferences of users through the observation of their actions. Those preferences are essential to obtain a satisfactory accuracy in the recommendations. Preference learning is especially difficult when attributes of different kinds (numeric or linguistic) intervene in the problem, and even more when they take multiple possible values. This paper presents an approach to learn user preferences over numeric and multi-valued linguistic attributes through the analysis of the user selections. The learning algorithm has been tested with real data on restaurants, showing a very good performance.

1 Introduction

Nowadays it is practically unconceivable to select our summer holiday destination or to choose which film to see in the cinema this weekend without consulting specialized sources of information in which, in some way or another, our preferences can be specified to aid the system to recommend us the best choices. That is because we live in an era where there are so many data easily available that it is impossible to manually filter every piece of information and evaluate it accurately. *Recommender Systems* (RS) have been designed to do this time-consuming task for us and, by feeding them with information about our interests, they are capable enough to tell us the best alternatives for us in a personalized way.

A RS stores the preferences of the user about the values of some criteria and uses this information to rate and sort a corpus of alternatives. The management of the preferences, the accuracy of the recommendations, and how these interests evolve over time are three of the most challenging tasks of these type of systems [8].

Concerning the first goal, RSs may obtain feedback from a user implicitly, explicitly or combining both approaches. This paper discusses an unsupervised way to infer the user interests, which observes the user interaction and does not require any explicit information from him [4].

The criteria used to describe the alternatives may have different natures. Some works propose the use of ontologies to represent concepts within a hierarchy [2,9]. Other researchers use fuzzy logic techniques to deal with linguistic criteria [1,10], and many approaches only consider numerical criteria [5]. This paper considers the use of linguistic and numerical data, which permit a high degree of expressivity and can be applied in a wide range of domains.

The basic idea is to use the preferences to sort a set of alternatives, show this ordered list to the user, and observe his final selection. With this information, the preference learning algorithm is able to modify the user profile so that it captures better the user preferences and the next recommendation is more accurate.

The rest of the paper is organized as follows. Section 2 includes a brief explanation of the related work the authors conducted in the area of preference learning over linguistic and numeric attributes, explaining how the interests of users over certain attributes or criteria are managed and learned. Section 3 explains a new approach to manage categorical attributes when they can take multiple linguistic values in a single alternative. Section 4 describes how a more expressive function which defines the behaviour of the preference over numeric attributes can be automatically learned. In Section 5 the case study where our approach has been tested (restaurant recommendation) is explained, describing the data set used and the results obtained. Finally, Section 6 gives the main conclusions of the paper and identifies some lines of future research.

2 Preference learning over categorical and numerical attributes

When we face a decision problem in which we require the aid of a RS to help us make a choice, all of the possible alternatives to said problem are defined, in most of the cases, by the same attributes. In this work we focus only on categorical and numeric attributes. The following subsections explain how preferences over the two different kinds of attributes are expressed, how alternatives are evaluated and ranked, and how the user interests are learned and adapted from his selections.

2.1 Attributes and management of preferences

In a recent work ([7]) we proposed to represent the level of interest over categorical attributes by using a linguistic scale in which preference labels are defined as fuzzy sets representing values of preference such as “Very Low”, “Low”, “Medium”, “High” or “Very High” (see Figure 1).

[1] Department of Computer Science and Mathematics, Universitat Rovira i Virgili, Tarragona, Catalonia (Spain). Email addresses: lucas.marin@urv.cat, antonio.moreno@urv.cat, david.isern@urv.cat.

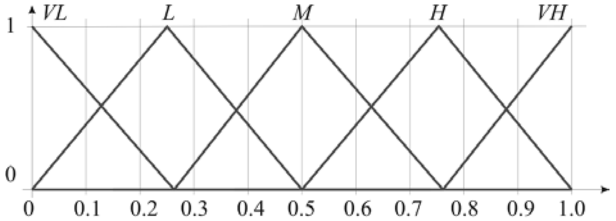


Figure 1. Example of a linguistic preference set

For the case of numeric attributes, we assumed that each user has a preference function for each attribute. This function has a triangular shape (see) and is defined as

$$p_a(x) = 1 - \frac{\left| x - v_{pref} \right|}{\Delta} \tag{1}$$

where $p_a(x)$ is the preference of the value x of the attribute a, and Δ is the width of the function, which we considered to be 10% of the attribute domain.

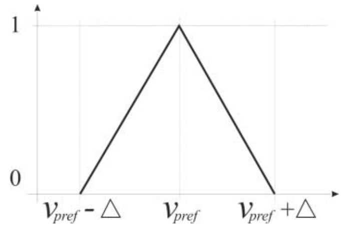


Figure 2. Basic numeric preference function

2.2 Alternatives evaluation

When evaluating an alternative, the objective is to aggregate all of the values of all of the attributes into a single value. Since we have two kinds of attributes, a conversion to the same domain is made. In our approach, we chose to translate the numerical preferences to linguistic ones. The translation is done by, first, calculating the value of preference of a certain numeric attribute value by using Eq. (1). Then that value is mapped to the fuzzy linguistic labels domain and matched with the label with a higher value in that point.

When all the attributes have been assigned a value of preference using the same fuzzy linguistic scale, all the terms are aggregated using the ULOWA aggregation operator [3]. The final result of this aggregation is the value of preference assigned to the whole alternative, used to rank the alternatives.

2.3 Preference learning

When the ranked alternatives are presented to the user, two things can happen: (a) the user selects the first ranked alternative or (b) the user selects any other alternative. The first case means that the recommendation process has worked accurately, since the system gave the first place to the selected alternative. However, in the second case, there were other alternatives (which we call *over ranked*) that were considered by the system as better than the one the user finally selected. Thus, that is probably indicating that the information that we have in the user profile is not accurate enough and should be modified. In a nut shell, the main intuition behind the user profile change algorithm is that we should increase the preference on the attribute values present in the selected alternative and decrease the preference on the attribute values appearing in the over ranked alternatives.

The information required to infer this reasoning is extracted from what is called “relevance feedback”. In this case, it consists in the over ranked alternatives and the selected one. Numerical and categorical attributes are managed in different ways, as described in the following subsections.

2.3.1 Linguistic preference adaptation

The main idea is to find attribute values repeated among the over ranked alternatives that do not appear on the selection, which will be the candidates for having his preference decreased. Similarly, the preference of the attribute values that appear on the selection and do not appear often on the over ranked alternatives is likely to be increased. The interested reader may find a more detailed explanation of the process of adaptation of linguistic preferences in [7].

The profile adaptation is conducted by two processes. The first one—called *on-line* adaptation—is executed every time the user asks the system for a recommendation, and it evaluates the information that can be extracted from the current ranked set of alternatives. The main goals of this stage are to *decrease* the preference of the attribute values that are causing non-desired alternatives to be given high scores and to *increase* the preference of the attribute values that are important for the user but are not well judged on the basis of the current user profile. For each recommendation made by the system, two sources of information are evaluated: the selected alternative, which is the choice made by the user, and the alternatives that were ranked above it. Values extracted from the over-ranked alternatives haver their level of preference *decreased* whereas the ones extracted from the user’s final selection that do not appear in the set of over-ranked alternatives have their preference *increased*.

The second one—called *off-line* adaptation—is triggered after the recommender system has been used a certain number of times. It considers the information given by the history of the previous rankings of alternatives and the selections made by the user in each case, but considers that information separately. When the system faces cases in which the number of over ranked alternatives is not large enough for reliable characteristics to be extracted, it stores the small number of over ranked alternatives in a temporary buffer. After several iterations in which the number of over ranked alternatives has been insufficient for evaluation, the system will have recorded enough alternatives to start evaluating them. When there are enough saved over-ranked alternatives, the values in their attributes will be analysed and their preference *decreased*. Moreover, user selections are also stored, and after a certain number of choices have been made, they are evaluated with the objective to increase the preference of the most repeated attribute values, since their repeated selection indicates that the user is really interested in them.

2.3.2 Numeric preference learning

The numeric adaptation of the user profile presented in [6] is inspired by Coulomb's Law: "*the magnitude of the electrostatics force of interaction between two point charges is directly proportional to the scalar multiplication of the magnitudes of charges and inversely proportional to the square of the distances between them*". The main idea is to consider the value stored in the profile (current preference) as a charge with the same polarity as the values of the same criterion on the over ranked alternatives, and with opposite polarity to the value of that criterion in the selected alternative. Thus, the value of the profile is pushed away by the values in the over ranked alternatives and pulled back by the value in the selected alternative. Two stages have been considered in the adaptation algorithm. The first one, called on-line adaptation process, is performed each time the user asks for a recommendation. The other stage, called off-line process, is performed after a certain amount of interactions with the user.

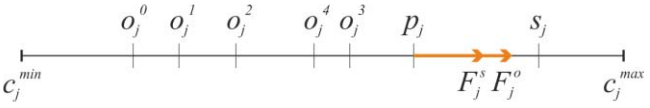


Figure 3. Attraction and repulsion forces

For the on-line stage, the information available in each iteration is the user selection and the set of over ranked alternatives. In order to calculate the change of the value of preference in the user profile for each criterion it is necessary to study the attraction force done by the selected alternative and the repulsion forces done by the over ranked ones in each criterion, as represented in the example in Figure 3, in which the *j*-th value of the five over ranked alternatives o^0, o^1, o^2, o^3, and o^4 causes a repulsion force $F^o{}_j$, and the value for the same criterion of the selected alternative, s_j, causes an attraction force $F^s{}_j$. Both forces are applied on the *j*-th value of the profile, P_j.

The attraction force F^s done by the selected alternative for each attribute *j* is defined as

$$F^s(P,s,j,\alpha)=\begin{cases}\Delta_j\left(\dfrac{1}{\left|s_j-P_j\right|^{\alpha}}\cdot\dfrac{s_j-P_j}{\left|s_j-P_j\right|}\right) & if\, s_j\neq P_j\\ 0 & if\, s_j=P_j\end{cases}\qquad(2)$$

In this equation, Δ_j is the range of the criterion *j*, s_j is the value of the criterion *j* in the selected alternative and P_j is the value of the same criterion in the stored profile *P*. The parameter α adjusts the strength of the force in order to have a balanced adaptation process. The repulsion force exerted by the over ranked alternatives for each criterion *j* is defined as a generalization of Eq.(2) as follows:

$$F^o\left(P,\left\{o^1,...,o^{no}\right\},j,\alpha\right)=\sum_{i=1}^{no}\frac{1}{\left|P_j-o_j^i\right|^{\alpha}}\cdot\frac{P_j-o_j^i}{\left|P_j-o_j^i\right|}\qquad(3)$$

Finally, both forces are summed up and the resulting force is calculated.

The techniques designed for the on-line stage fail at detecting user trends over time since they only have information of a single selection. The off-line adaptation process gathers information from several user interactions. This technique allows considering changes in the profile that have a higher reliability than those proposed by the on-line adaptation process, because they are supported by a larger set of data.

The off-line adaptation process can be triggered in two ways: the first one evaluates the user choices, while the second one analyses the over ranked alternatives discarded by the user in several iterations. The possibility of running the off-line process (in any of its two possible forms) is checked after each recommendation. In the first case, the system has collected some alternatives selected by the user in several recommendation steps, and it calculates the attraction forces (*F'*$_s$) exerted by each of the stored selected alternatives over the values stored in the profile, using an adaptation of Eq. (2), that has as inputs the profile *P*, the past selections {s^1,…,s^{rs}}, the criterion to evaluate *j*, and the strength-adjusting parameter α.:

$$F_s^{'}\left(P,\left\{s^1,...,s^{rs}\right\},j,\alpha\right)=\sum_{i=1}^{rs}\frac{1}{\left|s_j^i-P_j\right|^{\alpha}}\cdot\frac{s_j^i-P_j}{\left|s_j^i-P_j\right|}\qquad(4)$$

The second kind of off-line adaptation process evaluates the set of over ranked alternatives that have been collected through several iterations and which were not used in the on-line adaptation process (because it did not have enough over ranked alternatives in a single iteration). When the stored over ranked alternatives reach a certain number, the off-line adaptation process calculates the repulsion forces over the profile values exerted by those alternatives (F^o), which are calculated using Eq.(3).

3 Multi-valued linguistic attributes

As explained in Section 2, in our previous work we considered categorical attributes that could take only one linguistic value (For example, a city could have a single value in the "Climate" attribute). However, there are cases in which it is interesting to consider multiple values. One example of that situation could be the attribute "Types of food" in a restaurant: a restaurant can have the values {"Asian", "Seafood", "Vegetarian"} while another can have only "Italian food".

Extending our model so that it can manage lists of categorical values implies addressing two issues: how to represent and calculate the user preferences over the attribute taking into account all of the values and how to adapt dynamically those preferences.

3.1 Preference value on multi-valued attributes

When there is an attribute with multiple values a procedure should be defined to decide which single linguistic preference represents better the whole set of linguistic values.

Going back to the restaurant example, if a user has a "High" preference over "Asian food" restaurants and a "Low" preference over "Rice dishes", we can argue that the preference we could assign to the "Type of food" attribute in a restaurant with both values should be "Medium" (an average of the two kinds). If another restaurant only offers "Asian food" then its preference should be "High", so this restaurant would have a higher ranking than the first one. The rationale of this procedure is that it seems more adequate to reward the alternatives that are more focused in the aspects the user really likes. This example represents an "average" preference aggregation policy, however, other policies can also be considered depending on the attribute definition.

3.2 Preference learning on multi-valued categorical attributes

The linguistic algorithm used to adapt categorical preferences explained in Section 2 needs some improvements to be able to manage lists of values. When single-valued attributes were considered, the user selection pointed directly towards the value the user liked for that attribute. Now, however, we cannot be sure which one/s of the values listed in the attribute is/are the one/s of interest for the user. That is the reason why it has been necessary to design a "*relevance function*" which indicates how relevant is a value found among the over ranked alternatives or in the selected alternative. Relevance is measured in a [0,1] scale, with 1 meaning maximum relevance. To calculate how relevant a term *t* of the attribute *j* is among the over ranked alternatives we use this expression (the relevance value is 0 if it does not appear in the over ranked alternatives):

$$R_j^o(t) = \frac{1}{no}\sum_{i=1}^{nt}\frac{1}{nv_j^i} \qquad (5)$$

Here, *no* represents the number of over ranked alternatives, *nt* the number of over ranked alternatives where *t* appears, and nv^i_j the number of values that appear for the attribute *j* in the alternative *i*. In this equation we consider that every linguistic term that appears in the over ranked alternatives has a relevance which is inversely proportional to the number of other values for the same attribute that appear among the entire set of over ranked alternatives.

To calculate the relevance of a term in the selection we use:

$$R_j^s(t) = \frac{1}{2}\left(\frac{1}{nv_j} + \frac{nl}{tv}\right) \qquad (6)$$

Here nv_j represents the number of values that appear for the attribute *j* in the selection, *nl* the total number of linguistic attributes, and *tv* the total number of linguistic values that appear in the selection. The relevance of a term in the selection is the mean between the importance of the term among the values that appear with it in the same attribute and the importance of each linguistic term that appears in the selection compared with the number of linguistic attributes.

Finally, after calculating both partial relevancies for all the terms, the overall relevance $R_j(t)$ is calculated as:

$$R_j(t) = R_j^s(t) - R_j^o(t) \qquad (7)$$

In conclusion, considering a threshold *γ* to avoid making changes in the profile with low relevance, it can be deduced that:

- If $R_j(t)>\gamma$, the preference over term *t* for the attribute *j* needs to be increased (moved to the next term).
- If $R_j(t)<\gamma$, the preference over term *t* for the attribute *j* needs to be decreased (moved to the previous term).

4 Learning preference functions for numeric attributes

Although the numeric preference learning approach described in Section 2 provided an adequate way of learning the ideal value of preference over a numeric attribute, it was unable to learn all of the parameters that model the preference function such as the slope or the width, which were fixed. The new learning method presented in this section relies on historic data about the user selections to approximate the preference function of the numeric attributes to the most adequate one. With this approach, we have a new definition of the function of preference which now has 5 parameters (left and right slope, left and right width, and value of preference) instead of just the value of preference:

$$p_a(x)\begin{cases} 1-\dfrac{\left|x-v_{pref}\right|^{m_l}}{\Delta_l} & if\ (x<v_{pref}) \\ 1 & if\ (x=v_{pref}) \\ 1-\dfrac{\left|x-v_{pref}\right|^{m_r}}{\Delta_r} & if\ (x>v_{pref}) \end{cases} \qquad (8)$$

In this expression $p_a(x)$ is the preference of the value *x* of the attribute *a*, m_l and m_r are the function slope values (for the left and right sides of the triangle, respectively) and Δ_l and Δ_r are the parameters which define the width of the function (also for the left and right sides of the triangle, respectively). An example of graphical representation of a preference function can be seen in Figure 4, where the left slope is a value under 1, the right slope is a value over 1, and the left width is greater than the right one.

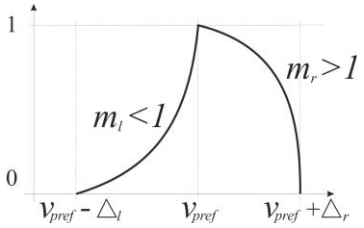


Figure 4. Numeric preference function with 5 parameters

The whole process of adapting the numeric preference function is depicted in Figure 5.

```
function PREF-FUNC-ADAPTATION(
    V(v_0,…,v_n), //historic of values of past selections
    v_pref, //value of maximum preference
    v_min, //minimum numeric value
    v_max, //maximum numeric value
    ti, //trust interval
    s //probability distribution sampling)
begin
    B=getBestValues(V, v_pref, ti);
    PD=calculateProbabilityDistribution(B, v_min, v_min, s);
    Δ{left,right}=calculateDelta(PD);
    m{left,right}=calculateBestSlope(PD, v_pref, Δ);
    PreferenceFunction=(Δ, m, v_pref);
    return PreferenceFunction;
end;
```

Figure 5. Preference function learning algorithm

The first step consists in obtaining the more reliable values from the historic set of selections. This is done by extracting a percentage of the values closer to the value of preference (trust interval), normally of 90%. With that we avoid considering outlier values. Then a probability distribution function, represented with a histogram, is calculated with those best values. The sample or discretization step is a parameter, normally around 1% of the domain range. Delta values are then calculated by observing the width of the probability distribution. For example, if the first value different to 0 in the histogram is 3 and the last is 56, and the value of higher preference (v_{pref}) is 34, Δ_l would be 31 and Δ_r would be 22. Afterwards, the algorithm generates preference functions with different combinations of values for the slope values (*m*) (in the range from 0 to 4 in steps of 0.2), and compares the distance between each preference function and the probability distribution. The function with the lower distance shows the chosen slope. Finally, the new preference function is built with the new delta and slope values.

5 Case study: restaurant recommendation

In order to test our new approach to multi-valued attribute evaluation and numeric preference function learning, we have used data of the restaurants in Barcelona to implement a RS with the ability to learn the users' interests from their selections. In the first part of this section a description of the data is given. Then, a basic explanation of the whole recommender and learning algorithm is given, as well as the preferences setup. Finally, the results of the evaluation are provided.

5.1 Barcelona restaurants data

The data used in this problem has been collected from the *BcnRestaurantes* web page[2]. The data set contains information about 3000 restaurants of Barcelona evaluated by 5 attributes: 3 categorical ("Type of food"- 15 values, "Atmosphere"- 14 values, "Special characteristics" – 12 values) and 2 numerical ("Average price", "Distance to city center"). One example of register in the data file is "*Fonda España; National, Season cuisine, Traditional; Classic, For families; Round tables, In a hotel, With video; 45; 0.979*", being "Fonda España" the restaurant name, "National", "Season cuisine" and "Traditional" the types of food served, "Classic" and "For families" the restaurant atmosphere, "Round tables" and "In a hotel" other important restaurant characteristics, 45€ the average menu price, and 0.979 km the distance to the city centre.

5.2 Recommendation and adaptation

The set of 3000 restaurants has been divided in blocks of 15 alternatives that are ranked independently, which gives out a total of 200 different recommendations. An ideal profile was manually defined and three initial profiles were created randomly. The goal is to learn the ideal profile starting from these three different points. In this evaluation the preferences over the categorical attributes are represented with a linguistic label term set of 7 values, which are "Very Low", "Low", "Almost Low", "Medium", "Almost High", "High" and "Very High".

The whole process (for each of the three profiles, repeated 200 times) consists in:

1. Ranking a set of 15 alternatives according the current (initially random) profile.
2. Simulate the selection of the user by choosing the alternative that fits better the ideal profile.
3. Extract relevance feedback from the selection (over ranked alternatives and the selection itself).
4. Decide which changes need to be made to the current profile and apply them.

Some information about the whole process is stored after each iteration, including the position of the selected alternative, the distance between the ideal and current profiles, and the preferences over linguistic and numeric values.

5.3 Results evaluation

In order to evaluate the results of the new learning techniques, a distance function has been defined to calculate how different the profile we are learning is to an ideal profile which represents the exact preferences of the user. The first step is to calculate the distance for each attribute, taking into account if it is numeric or categorical. The distance between numeric attributes is calculated as

$$d(n,c,i) = 1 - p_n^c(v_{pref\,n}^{\;i}) \tag{9}$$

where *n* is the numerical attribute, *c* is the current profile (the one being learned), *i* is the ideal profile, and $p_n^c(v_{pref\,n}^{\;i})$ is the value of preference of the v_{pref} value for the attribute *n* in *i* using the preference function of the same attribute in the profile *c*. A distance 0 means that the v_{pref} values in both profiles are equal.

The equation to calculate the distance between categorical attributes is

$$d(l,c,i) = \frac{1}{card(l)} \sum_{k=1}^{card(l)} \frac{\left|CoG(p_l^c(v_k)) - CoG(p_l^i(v_k))\right|}{\left|CoG(s_{\min}) - CoG(s_{\max})\right|} \tag{10}$$

where *l* is the categorical attribute, *card(l)* is the cardinality of the attribute *l* (i.e., the number of different linguistic values it can take), $CoG(p_l^c(v_k))$ and $CoG(p_l^i(v_k))$ are the x-coordinate of the centres of gravity of the fuzzy linguistic labels associated to the value of preference of v_k in the profiles *c* and *i*, respectively, and $CoG(s_{\min})$ and $CoG(s_{\max})$ are the centres of gravity of the minimum and maximum labels of the domain, respectively. Finally, the distance between two profiles is calculated as

$$D(c,i) = \frac{1}{na} \sum_{k=1}^{na} d(k,c,i) \tag{11}$$

where *na* is the total number of attributes.

During the three tests (one for each initial random profile) the distance between the adapting and the ideal profile has been calculated in each iteration. Figure 6 (continuous line) shows the average of the three distances. It can be seen that the initial average distance between the ideal and the adapting profiles is around 0.59. After 200 iterations it reaches a distance around 0.1. Although 200 iterations may seem a large number, it can also be observed that with only 50 iterations a very acceptable result of 0.2 is obtained.

To see to what extent the new approach to learn the numeric preference function explained in Section 4 has improved the result of our previous work (commented in Section 2), Figure 6 also compares the results with and without (dashed line) that functionality. It can be seen how the improvement has been noticeable (distance improvement of about 0.07).

[2] http://www.bcnrestaurantes.com. Last access May 30th, 2012.

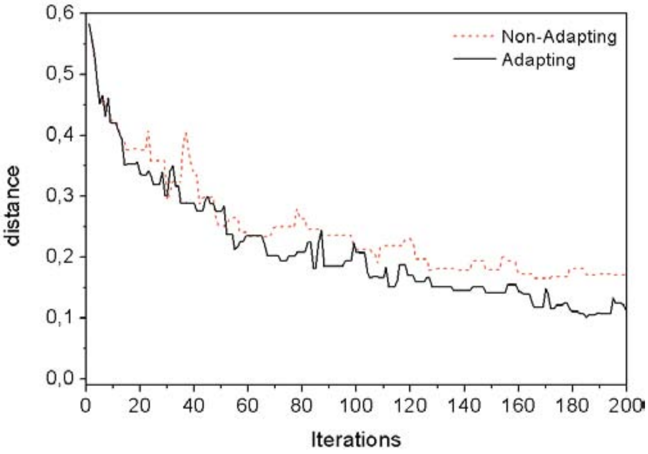


Figure 6. Average distance between current and ideal profile

To wrap up the results evaluation, Figure 7 shows in what position the user selection is being ranked by the RS on each of the iterations in the first test (the three give similar results). This figure shows the results in a more intuitive way. Notice that the system is accurate if the selected alternative is in the first positions of the 15-items list in each iteration. Many factors can interfere in the process and make the learning of the exact ideal profile a very hard task, but if the user selection appears in the first positions, we can consider that the learning process is working properly. As it can be observed in Fig.7, after about 50 iterations, the selected alternative is among the first three ones in 95% of the cases (and the first one in around 70% of the cases).

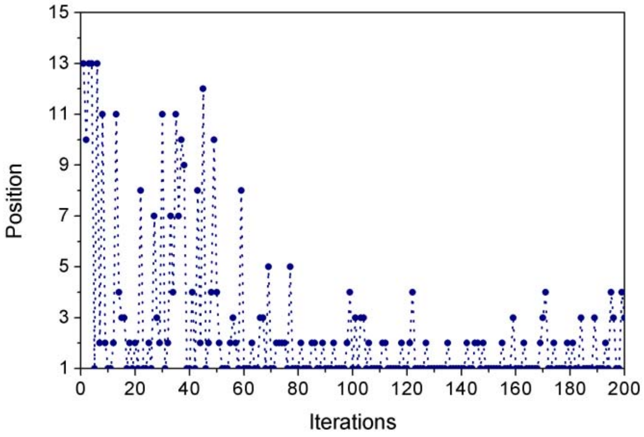


Figure 7. Position of the selected alternative in each iteration (test 1)

6 Conclusions and future work

Two main contributions with respect to our previous work have been presented in this paper. The first one consists in managing multi-valued categorical attributes in the alternatives of a RS, allowing. more expressivity in their representation. The system considers a single preference for each possible value and aggregates them to find out the preference over the whole attribute. The consideration of multi-valued attributes is mandatory when working with alternatives such as the ones presented in this paper (e.g. “Type(s) of Food” in a restaurant alternative).

The second contribution, which is learning the numeric preference function, allows shaping a more expressive and personalised representation of user preferences over each numeric attribute, defining a preference function with 5 parameters. This additional expressivity helped to improve the profile learning process by reducing the learning error around 7%.

As a future work, two interesting lines can be considered. As pointed out in Section 3, an aggregation policy can be considered in the aggregation of the preferences in a single attribute, other than the use of the common “average” policy. Research can be made in this area in order to learn the aggregation policy that fits more the user interests. Another interesting line to consider is to incorporate information about the numeric preference function in the distance measure used to evaluate the algorithm since, currently, just the value of preference is being considered.

ACKNOWLEDGEMENTS

This work has been supported by the Universitat Rovira i Virgili (a pre-doctoral grant of L. Marin) and the Spanish Ministry of Science and Innovation (DAMASK project, Data mining algorithms with semantic knowledge, TIN2009-11005) and the Spanish Government (Plan E, Spanish Economy and Employment Stimulation Plan).

REFERENCES

[1] G. Castellano, C. Castiello, D. Dell'Agnello, A. M. Fanelli, C. Mencar, M. A. Torsello, Learning Fuzzy User Profiles for Resource Recommendation, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 18 (4) (2010), 389-410.

[2] I. Garcia, L. Sebastia, E. Onaindia, On the design of individual and group recommender systems for tourism, Expert Systems with Applications 38 (6) (2011), 7683-7692.

[3] D. Isern, L. Marin, A. Valls, A. Moreno, The Unbalanced Linguistic Ordered Weighted Averaging Operator, in: IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2010, IEEE Computer Society, Barcelona, Catalonia, 2010, 3063-3070.

[4] G. Jawaheer, M. Szomszor, P. Kostkova, Comparison of Implicit and Explicit Feedback from an Online Music Recommendation Service, in: 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems, ACM Press, Chicago, US, 2010, 47-51.

[5] T. Joachims, F. Radlinski, Search Engines that Learn from Implicit Feedback, Computer 40 (8) (2007), 34-40.

[6] L. Marin, D. Isern, A. Moreno, Dynamic adaptation of numerical attributes in a user profile, International Journal of Innovative Computing Information and Control (2012).

[7] L. Marin, D. Isern, A. Moreno, A. Valls, On-line dynamic adaptation of fuzzy preferences, Information Sciences doi: 10.1016/j.ins.2011.10.008 (2012).

[8] M. Montaner, B. López, J. L. de La Rosa, A taxonomy of recommender agents on the internet, Artificial Intelligence Review 19 (4) (2003), 285-330.

[9] A. Moreno, A. Valls, D. Isern, L. Marin, J. Borràs, SigTur/E-Destination: Ontology-based personalized recommendation of Tourism and Leisure Activities, Engineering Applications of Artificial Intelligence doi:10.1016/j.engappai.2012.02.014 (2012).

[10] C. Porcel, A. G. López-Herrera, E. Herrera-Viedma, A recommender system for research resources based on fuzzy linguistic modeling, Expert Systems with Applications 36 (3, Part 1) (2009), 5173-5183.

Direct Value Learning: a Preference-based Approach to Reinforcement Learning

David Meunier$^{(1)}$**, Yutaka Deguchi**$^{(2)}$**, Riad Akrour**$^{(1)}$
Einoshin Suzuki$^{(2)}$**, Marc Schoenauer**$^{(1)}$**, Michele Sebag**$^{(1)}$

Abstract. Learning by imitation, among the most promising techniques for reinforcement learning in complex domains, critically depends on the human designer ability to provide sufficiently many demonstrations of satisfactory quality.

The approach presented in this paper, referred to as DIVA (Direct Value Learning for Reinforcement Learning), aims at addressing both above limitations by exploiting simple experiments. The approach stems from a straightforward remark: while it is rather easy to set a robot in a target situation, the quality of its situation will naturally deteriorate upon the action of naive controllers. The demonstration of such naive controllers can thus be used to learn directly a value function, through a preference learning approach. Under some conditions on the transition model, this value function enables to define an optimal controller.

The DIVA approach is experimentally demonstrated by teaching a robot to follow another robot. Importantly, the approach does not require any robotic simulator to be available, nor does it require any pattern-recognition primitive (e.g. seeing the other robot) to be provided.

1 Introduction

Since the early 2000s, significant advances in reinforcement learning (RL) have been made through using direct expert's input (inverse reinforcement learning [18], learning by imitation [10], learning by demonstration [16]), assuming the expert's ability to demonstrate quasi-optimal behaviors and to provide an informed representation.

In 2011, new RL settings based on preference learning and allegedly less demanding for the expert have been proposed (more in section 2).

In this paper, a new preference-based reinforcement learning approach called DIVA (*Direct Value Learning for Reinforcement Learning*) is proposed. DIVA aims at learning directly the value function from basic experiments. The approach is illustrated on the simple problem of teaching a robot to follow another robot. It is shown that DIVA yields a competent follower controller without requiring the primitive "I see the other robot" to be either provided, or explicitly learned.

2 State of the art

Reinforcement learning is most generally formalized as a Markov decision process. It involves a state space $\mathcal{S}$, an action space $\mathcal{A}$, and an upper bounded reward function r defined on the state space $r : \mathcal{S} \mapsto \mathbb{R}$. The model of the world is given by the transition function $p(s, a, s')$, expressing the probability of arriving in state s' on making action a in state s under the Markov property; in the deterministic case, the transition function $tr : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ gives the state $tr(s, a)$ of the agent upon making action a in state s. A policy $\pi : (\mathcal{S}, \mathcal{A}) \mapsto \mathbb{R}$ maps each state in $\mathcal{S}$ on some action in $\mathcal{A}$ with a given probability. The return of policy π is defined as the expectation of cumulative reward gathered along time when selecting the current action after π, where the initial state s_0 is drawn after some probability distribution q on $\mathcal{S}$. Denoting $a_h \sim \pi(s_h)$ the random variable action selected by π in state s_h, $s_{h+1} \sim p(s_h, a_h, s)$ the state of the agent at step $h + 1$ conditionally to being in state s_h and selecting action a_h at step h, and r_{h+1} the reward collected in s_{h+1}, then the policy return is

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{h=0}^{\infty} \gamma^h r_h | s_0 \sim q \right]$$

where $\gamma < 1$ is a discount factor enforcing the boundedness of the return, and favoring the reaping of rewards as early as possibly in the agent lifetime.

The so-called value function $V_\pi(s)$ estimates the expectation of the cumulative reward gathered by policy π when starting in state s, recursively given as:

$$V_\pi(s) = r(s) + \gamma \sum_{a,s'} \pi(s, a) p(s, a, s') V_\pi(s')$$

Interestingly, from a value function V can be derived a greedy policy π_V, provided the transition function is known: when in state s, select the action a leading to the state with best expected value:

$$\pi_V(s) = \text{ argmax }_{a \in \mathcal{A}} \left\{ \begin{array}{ll} p(s, a, s')V(s') & \text{probabilistic trans.} \\ V(tr(s, a)) & \text{deterministic trans.} \end{array} \right\} \tag{1}$$

By construction, π_{V_π} is bound to improve on π. By learning value function V^* as the maximum over all policies π of V_π

$$V^*(s) = max_\pi V_\pi(s)$$

one can thus derive the optimal policy $\pi^* = \pi_{V^*}$.

The interested reader is referred to [19, 20] for a comprehensive presentation of the main approaches to Reinforcement Learning, namely value iteration and policy iteration algorithms, building a sequence of value functions V^i and policies π_i converging to V^* and π^*. The bottleneck of estimating the optimal value function is that all states must be visited sufficiently often, and all actions must be triggered in any state, in order to enforce the convergence of V^i and $\pi(V^i)$ toward V^* and π^*. For this reason, RL algorithms hardly scale up when the size of the state and action spaces is large, all the more so as the state description must encapsulate every information relevant to action selection in order to enforce the Markov property.

New RL approaches devised to alleviate this bottleneck and based on preference learning have been proposed in 2011 [11, 2]. [11] is concerned with the design of the reward function in order to facilitate RL; for instance in the medical protocol application domains,

[1] TAO, CNRS-INRIA-LRI, Université Paris-Sud
[2] Dept. Informatics, ISEE, Kyushu University

how to associate a numerical negative reward to the patient's death ? The authors thus extend the classification-oriented approach first proposed by [17] as follows. In a given state s, an action a is assessed by executing the current policy until reaching a terminal state (rollout). On the basis of these assessments, pairs of actions can be ranked with regard to state s and policy π ($a <_{s,\pi} a'$). These ranking constraints are exploited within a learning-to-rank algorithm (e.g. RankSVM [13]), yielding a ranking hypothesis $h_{s,\pi} : \mathcal{A} \mapsto \mathbb{R}$. The claim is that such action ranking hypotheses are more flexible than classification hypotheses, aimed at discriminating "the" best actions from the others conditioned by the current state. In summary, the ranking hypothesis depends on the policy and the current state, and operates on the action space.

Quite the contrary, in [2] the ranking hypothesis operates on the policy space. The motivating application is swarm robotics, facing two severe issues. Firstly, swarm robotics is hardly compatible with generative model-based RL approaches; simulator-based approaches suffer from the supra-linear computational complexity of simulations w.r.t. the number of robots in the swarm (besides the simulation noise). Secondly, swarm robotics hinders the inverse reinforcement learning approach [1, 15], using the expert demonstrations to learn a reward function. In most cases the swarm expert cannot describe (let alone demonstrate) the individual robot behavior leading to the desired swarm behavior (known as the inverse macro-micro problem [7]). The proposed approach, called PPL (*Preference-based Policy Learning*) proceeds along an interactive optimization setting: the robot(s) demonstrates a behavior, which is ranked by the expert comparatively to the previous best demonstration. The ranking constraints are exploited through a learning-to-rank algorithm, yielding a ranking hypothesis on the policy demonstrations and thus on the policy space Π ($h : \Pi \mapsto \mathbb{R}$). This ranking hypothesis is used as policy return estimate, casting RL as an optimization problem (find $\pi^*_h = \text{ argmax } h(\pi)$). Policy π^*_h is demonstrated to the expert, who ranks it compared to the previous best demonstration, and the process is iterated. Note that PPL thus faces the same difficulty as interactive optimization at large [9, 22]: if the expert is presented with too constrained a sample of demonstrations, she does not have a chance to teach her preferences to the system. PPL is thus extended to integrate an active learning criterion, yielding the APRIL (*Active Preference learning-based Reinforcement Learning*) algorithm [3].

3 Overview of DIVA

This section introduces and formalizes the principle of DIVA, and discusses its strengths and limitations w.r.t. the state of the art.

3.1 Principle

DIVA is rooted in Murphy's law (*Anything that can possibly go wrong, does*). Formally, it posits that when the agent happens to be in some good situation, its situation tends to deteriorate under *most* policies. Let us illustrate this idea on the simple problem of having a robot following another robot in an open environment. Assume that the follower robot is initially situated behind the leader robot (Fig. 1, left, depicts the follower state, given as its camera image). Assume that both follower and leader robots are equipped with the same simple *Go Ahead* controller (same actuator value on the left and right wheel of both robots). Almost surely, each robot trajectory will deviate from the straight line, due to e.g. the imperfect calibration of the wheel actuators or different sliding frictions on the ground. Almost surely, the two robot trajectories will be deviated in a different way. Therefore, the follower will at some point lose track of the leader (Fig. 1, right, depicts the follower state after circa 52.7 (+-20.6) time steps, that is, XX seconds).

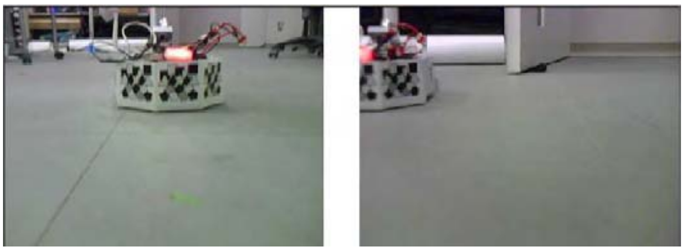


Figure 1: Left: The follower robot is initially aligned behind the leader robot. Right: Both leader and follower robots are operated by the same *Go ahead* controller. Due to mechanical drift, the follower sooner or later loses track of the leader.

The intuition can be summarized as: the follower state was never as good as in the initial time step; it becomes worse and worse along time.

3.2 Formalization

The above remarks enable to define a ranking hypothesis on the state space, as follows. Let us consider K trajectories of the robot follower noted $S_1 \ldots S_K$, where each trajectory S_i is defined as a sequence of states $s^{(i)}_t, t = 0 \ldots T_i$. A value function is sought as a function V mapping the set of states $\mathcal{S}$ onto $\mathbb{R}$, satisfying constraints $V(s^{(i)}_t) > V(s^{(i)}_{t+1})$ for all $i = 1 \ldots K$ and $t = 0 \ldots T_i - 1$.

Formally, it is assumed in the following that the state space $\mathcal{S}$ is embedded in $\mathbb{R}^d$. A linear value function $\widehat{V^*} : \mathcal{S} \mapsto \mathbb{R}$ is defined as

$$\widehat{V^*}(s) = \langle \widehat{w}^*, s \rangle$$

where $\widehat{w}^* \in \mathbb{R}^d$ is given after the standard learning-to-rank regularized formulation [4]:

$$\begin{array}{rl} \widehat{w}^* = & \text{arg min } \frac{1}{2}||w||^2_2 + C \sum_{i=1}^{k} \sum_{t<t' \leq T_i} \xi^{(i)}_{t,t'} \\ & s.t. \; \forall \, 1 \leq i \leq K, \; 0 \leq t < t' \leq T_i \\ & \langle w, s^{(i)}_t \rangle \geq \langle w, s^{(i)}_{t'} \rangle + 1 - \xi^{(i)}_{t,t'}; \qquad \xi^{(i)}_{t,t'} \geq 0 \end{array} \quad (2)$$

This quadratic optimization under constraints problem can be solved with affordable empirical complexity [14]. After Eq. (1) and provided that the transition model is known, value function $\widehat{V^*}$ derives a policy $\widehat{\pi}^*$. Further, by construction any value function derived by monotonous transformation of $\widehat{V^*}$ induces the same policy $\widehat{\pi}^*$.

3.3 Discussion

Among the main inspirations of the DIVA approach is TD-Gammon [21]. TD-Gammon, the first backgammon program to reach a champion level in the 80s, exploits games generated from self-play to train a value function along a temporal difference algorithm. The value function is likewise trained from a set of games, or trajectories S_i described as a sequence of positions $s^{(i)}_t, t = 0 \ldots T_i$. The difference is as follows. Firstly, TD-Gammon only imposes the value for the initial and the final positions, with $V(s^{(i)}_0) = 1/2$ (the initial position is neutral), and $V(s^{(i)}_{T_i}) = 1$ (respectively 0) if the first player wins (resp., loses) the game. Secondly, the learning problem is regularized through a total variation minimization (minimizing $\sum_i \sum_{t=1}^{T_i-1}(V(s_t^{(i)}) - V(s_{t+1}^{(i)}))^2$).

A key difference between DiVA and TD-Gammon regards the availability of a simulator. If a generative model (a simulator) is available for the problem domain, then indeed RL can rely on cheap and abundant training data. In robotics however, simulator-based training is prone to the so-called reality gap [8]. Generally speaking, the robotic framework is hardly compatible with data-intensive learning approaches, due to the poor accuracy of simulator-based data on the one hand, and the cost (experimenter time and robot fatigue) of in-situ experiments on the other hand.

A consequence is that TD-Gammon can exploit high quality simulated data whereas DiVA relies on a limited amount of data. Further, these data are experimentally acquired and should require little or no expertise from the human experimenter. Along the same line, TD-Gammon only prescribes the values attached to the initial and final state of each trajectory; the bulk of learning relies on the regularization term. Quite the contrary, DiVA exploits short trajectories and sets comparison constraints on the values attached to each time step (besides using regularization too). Finally, TD-Gammon requires all winning/losing terminal states to be associated the same value; there is no such constraint in DiVA, as states from different trajectories are not comparable.

Both approaches suffer from a same limitation: policy π_V is computed from value function V iff the transition function is known or well approximated (Eq. (1)). Further, finding the optimal action requires one to solve an optimization problem in each time step, which usually requires the action space to be small. It will be seen (section 4.2) that a cheap estimate of the transition function can alleviate the above limitations in the robotic framework in some cases. An additional limitation of DiVA is that the value function V is learned from a set of trajectories obtained from a "naive" controller, which does not necessarily reflect the target (test) distribution, thus raising a transfer learning problem [6]. This issue will be discussed in section 5.

4 Experimental validation

A proof of principle of DiVA is given in a robotic framework (section 4.1), based on a delayed transition model estimate (section 4.2) and a continuity assumption (section 4.3). The experimental setting and goals of experiments are described in section 4.4 and results are reported and discussed in section 4.5.

4.1 Framework

The robotic platform is a Pandaboard, driven by the dual-core ARM Cortex-A9 OMAP4430, with each core running at 1 GHz, and equipped with 1 GB DDR2 RAM. The robot is equipped with a USB camera with resolution (320×240), and color depth of monochrome 8bit. All experiments are done in-situ, taking place in a real laboratory full of tables, chairs, feet and various (moving) obstacles. Although no model of the world (transition function or simulator) is available, a cheap estimate thereof can be defined (see below).

The primary description of the robot state is given by its camera image (in $\mathbb{R}^{76800}$). A pre-processing step aimed at dimensionality reduction and inspired from the SIFT (scale-invariant feature transform) descriptors is used and operated on the Pandaboard using the Linux OS. More precisely, SURF (speeded up robust feature) descriptors are used, claimed to be several times faster and more robust against different image transformations than SIFT [5]. Finally, each block of $d \times d$ contiguous pixels in the initial image is represented by an integer, the number of SURFs occurring in the block ($d = 1, 2, 4, 16$ in the experiments, with $d = 4$ the best empirical setting, and the only considered in the remainder of the paper). The state space $\mathcal{S}$ is finally included in $\mathbb{R}^{4800}$.

4.2 Delayed transition model estimate

In the following, the action space $\mathcal{A}$ includes three actions: *Ahead*, *Right* and *Left*. The extension to richer and more gradual action spaces is left for further work. As already mentioned, the definition of a controller from a value function requires a transition model to be available (Eq. (1)). Two working assumptions are done to overcome the lack of an accurate transition model for the considered open world.

Firstly, a delayed transition model estimate is defined as follows. Let s_{t-1} and s_t denote the states at time $t-1$ and t respectively, and assume that the action selected at time $t-1$ is *Ahead*. Then, the idea is that if the robot had turned right at time $t-1$, it would have seen approximately the same image as in s_t, but translated on the left; additionally, some new information would have been recorded on the rightmost camera pixels while the information on the leftmost pixels would have been lost. In other words, given $s_t = tr(s_{t-1}, \textit{Ahead})$ one can compute an approximation of $tr(s_{t-1}, \textit{Right})$, by a circular shift of s_t. Let state s_t be described as a pixel matrix $s_t[i, j]$ where $i = 1 \ldots n_w$, $j = 1 \ldots n_h$ and n_w (respectively n_h) is the width (resp. height) of the camera image. Then,

$$tr(s_{t-1}, \textit{Right}) \approx \{s_t[i + \Delta \pmod{n_w}, j]\}$$

where Δ is a constant translation width (64 pixels in the experiments; this will be relaxed in further work, section 5). Note that the unknown rightmost pixels in $tr(s_{t-1}, \textit{Right})$ are arbitrarily replaced by the leftmost pixels in s_t; the impact of this approximation will be discussed further.

More generally, given $s_t = tr(s_{t-1}, a)$, a delayed transition model estimate is given by

$$\widehat{tr}(s_{t-1}, a'|a, s_t = tr(s_{t-1}, a)) = \{s_t[i + \ell(a, a') \pmod{n_w}, j]\}$$

where the translation width $\ell(a, a')$ is Δ (resp. $-\Delta$) if a= *Ahead*, a' =*Right* (resp. *Left*), and completed by consistency (Table 3).

a	a'	$\ell(a, a')$	a	a'	$\ell(a, a')$
Ahead	*Right*	Δ	*Ahead*	*Left*	$-\Delta$
Left	*Ahead*	Δ	*Right*	*Ahead*	$-\Delta$
Left	*Right*	2Δ	*Right*	*Left*	-2Δ

(3)

This estimate is delayed as it is only available at time t, since $\widehat{tr}(s_{t-1}, a'|a, s_t = tr(s_{t-1}, a))$ is computed from s_t. To some extent, this estimate can cope with partially observable environments (another robot of the swarm might arrive in sight of the current robot at t, while it was not seen at time $t-1$). On the other hand, the estimation error is known to be concentrated in the peripheral pixels.

4.3 Continuity assumption

Given a value function $\widehat{V^*}$, the action a_{t-1} which has been selected at $t-1$ and the current state s_t, the delayed transition model estimate defines *what would have been the best action* a^*_{t-1} that should have been selected at time $t-1$ instead of a_{t-1}, after Eq. (1):

$$a^*_{t-1} = argmax_{\;a \in \mathcal{A}} \left\{ \widehat{V^*}(\widehat{tr}(s_{t-1}, a|a_{t-1}, s_t)) \right\} \qquad (4)$$

Figure 2: A lesion study: recording irrelevant states at the beginning of the follower trajectory (the white board on the left and the author feet on the right).

The continuity assumption posits that the environment changes gracefully, implying that the action which was the most appropriate at time $t-1$ is still appropriate at time t. Along this line, the controller defined from $\widehat{V^*}$ and noted $\widehat{\pi}^*$ selects at time t action a^*_{t-1} as defined by Eq. (4). A further assumption behind the definition of $\widehat{\pi}^*$ is that the noise of the delayed transition model estimate is moderate with respect to $\widehat{V^*}$. In other words, it is assumed that the peripheral pixels (unknown in truth and arbitrarily filled from s_t) are not key to value function $\widehat{V^*}$ (the corresponding weights have low absolute value).

4.4 Experimental setting

The controller goal is to enable the follower robot to follow the leader robot.

Eleven trajectories are recorded. Each trajectory is initialized with the follower positioned behind the leader at various locations in the lab, and both robots operated with the constant policy $\pi_0(s) = Ahead$ for 128 time steps. The i-th trajectory thus records the follower state $s_t^{(i)}, t = 1 \ldots 128$. A computational time step amounts to circa .5 second of real-time (two frames per second), due to the on-board computation of the SURF descriptors. The value function $\widehat{V^*}$ is learned from these trajectories as in section 3.2.

The primary goal of experiments is to study the performance of controller $\widehat{\pi}^*$, in terms of the average time the follower can actually follow the leader. Further, in order to allow for larger time horizons, the controller run on the leader robot is an obstacle avoidance (Braitenberg) controller, enabling the leader to run for a couple of hundred time steps before being stopped. Note that modifying the leader policy amounts to considering a different environment, making the goal more challenging as the test setting differs from the training one: the leader can turn abruptly upon seeing an obstacle, whereas it turns only very gradually in the training trajectories.

The second goal of experiments is to assess the robustness of the approach with respect to noise. A lesion study is conducted by perturbing the initial states in the training trajectories, e.g. recording the images seen by the follower (e.g. the walls or the experimenter) before the follower actually starts to follow the leader (Fig 2). The value function learned from the perturbed trajectories and the associated controller are referred to as NOISY-DIVA.

The DIVA approach and the merits of a rank-based approach are also assessed by comparison with a simple regression based approach, referred to as REG-DIVA, where value function $\widehat{V^*}$ is regressed from the training set $\mathcal{E} = \{(s_t^{(i)}, -t), i = 1 \ldots 11, t = 1 \ldots 128\}$.

Finally, the assumption that peripheral pixels are not relevant to value function $\widehat{V^*}$ (section 4.3) is also examined, depicting the average weight of the value function for each (block of) pixel(s) in the image.

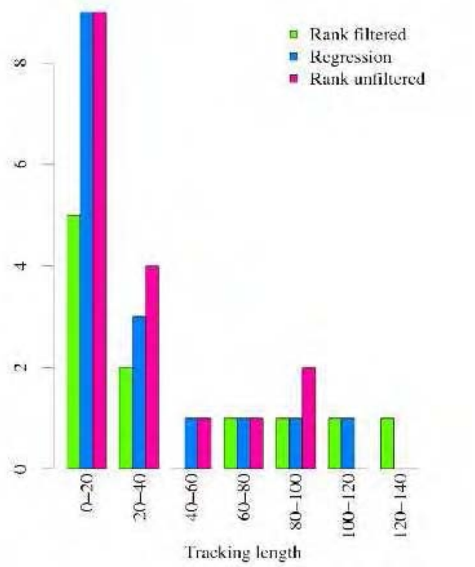


Figure 3: Comparative results of DIVA, NOISY-DIVA and REG-DIVA: Histogram of the number of consecutive time steps the follower keeps on following a Braitenberg-operated leader, over 5 runs. One time step corresponds to ca half a second, due to the on-board computation of the SURF descriptors (2 frames per second).

4.5 Results

The performance of the DIVA controller is assessed by learning $\widehat{V^*}$ from all 11 training trajectoriesin DIVA NOISY-DIVA and REG-DIVA modes and running the associated $\widehat{\pi}^*$ controllers. Every controller $\widehat{\pi}^*$ is launched on the follower robot for five runs. In each run, the follower is initially positioned behind the Braitenberg-operated leader, at various locations in the lab (same initial locations for DIVA, NOISY-DIVA and REG-DIVA settings). In each run, the follower is manually repositioned behind the leader when it loses the track.

The histogram over the 5 runs of the number of consecutive time steps while the follower does follow the leader is displayed in Fig. 3, reporting the respective performances of DIVA, NOISY-DIVA and REG-DIVA. As was expected, the best results are obtained for the noiseless rank-based DIVA approach, followed by the noisy rank-based NOISY-DIVA approach. The fact that some tracking sequences are very short is explained as the leader meets very soon an obstacle and turns fast, making the follower lose the track. Overall, the follower stays on the leader track for over 60 time steps in about 40% of the experiments while the track was lost after 52.7 (+-20.6) time steps in the training experiments.

The performance of the DIVA controller is also assessed on the training trajectories along a leave-one-out procedure, learning 11 value functions $\widehat{V^*}^{(i)}$ from all trajectories but the i-th one. $\widehat{V^*}^{(i)}$ is computed on the remaining i-th trajectory (Fig. 4). As expected, the value of $\widehat{V^*}^{(i)}$ decreases along time. Interestingly, DIVA and NOISY-DIVA exhibit similar behaviors, rapidly decreasing as t increases although the value remains high for some runs; after visual inspection, the drift occurs late for these runs. The behavior of REG-DIVA is more erratic, which is explained as the regression setting is over constrained regarding the quality of the experiments, requiring all states $s_t^{(i)}$ to be associated to the same value $-t$.

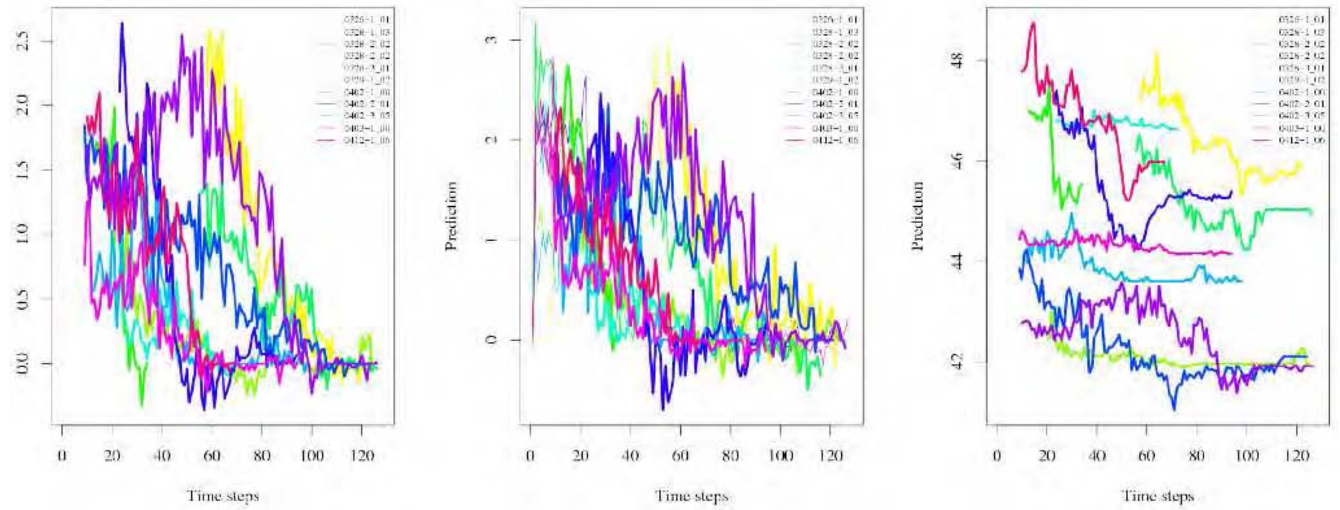


Figure 4: Comparative behavior of the value function $\widehat{V^*}_i$ along time. A leave-one-out procedure is used: the value function is learned from all but one trajectory, and its value on the remaining trajectory is reported. The value function learned by DIVA, NOISY-DIVA and REG-DIVA are reported respectively on the left, middle and right.

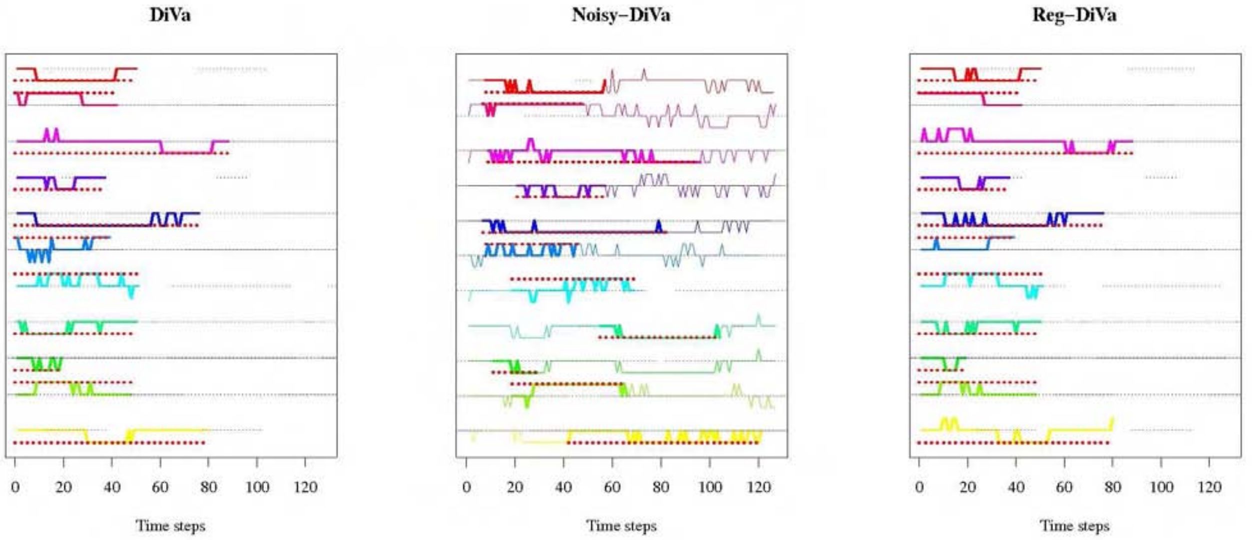


Figure 5: Comparative behavior of controller $\pi_{\widehat{V^*}_i}$ on the test trajectory along time, for DIVA (left), NOISY-DIVA (medium) and REG-DIVA (right).

The corresponding policy $\widehat{\pi}^{*(i)}$is plot in Fig. 5, where $\widehat{\pi}^{*(i)}(s_t^{(i)})$ is indicated by a cross respectively on, above or below over the central line depending on whether the selected action is *Ahead*, *Right* or *Left*. The ground truth obtained by visual inspection of the logs is reported below. It is seen that the policy generally selects the relevant action, and becomes chaotic in the end of the trajectory as the follower does no longer see the leader.

Finally, the working assumption that peripheral pixels hardly matter for the value function (section 4.3), implying that the noise of the delayed transition model estimate does not harm the controller, is confirmed by inspecting the average weight of the pixels in Fig. 6. As could have been expected, the most important region is the central upper one, where the leader is seen at the beginning of each training trajectory; the low region overall is considered uninformative. In the upper regions, the weight decreases from the center to the left and right boundaries.

5 Discussion and perspectives

This paper has presented a proof of principle of the DIVA approach, showing that elementary experiments can be used to "prime the pump" of reinforcement learning and train a mildly competent value function. Indeed, a controller with comparable performance might have been manually written (hacked) easily. Many a student experience suggests however that the manual approach is subject to the law of diminishing returns, as more and more efforts are required to improve the controller as its performance increases.

The next step thus is to study the scalability of DIVA, e.g. by gradually adding new logs to the training set. On-going experiments consider the effects of adding the test trajectories (with Braitenberg-

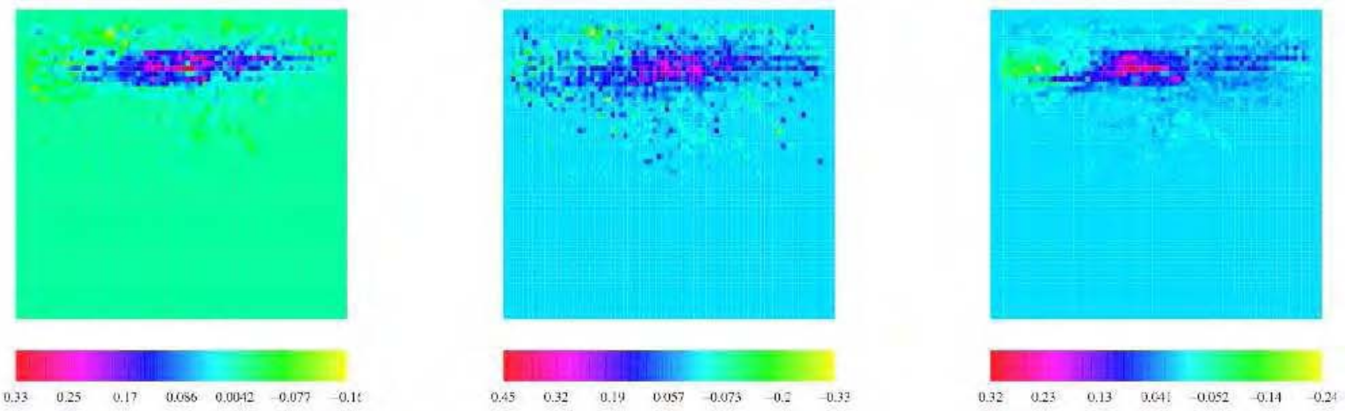


Figure 6: Average relevance of the visual regions to the value function for DIVA (left), NOISY-DIVA (medium) and REG-DIVA (right).

operated leader) to the training trajectories to retrain the value function.

The main current limitation of the approach concerns the small size of the action space. Still, it must be noted that the delayed transition model estimate naturally extends to fine-grained actions, by mapping the group of movement actions on the group of image translation vectors (section 4.2), thus enabling to determine the optimal action in a large action space. For computational efficiency, the underlying requirement is that the target behavior be sufficiently smooth. Currently, this requirement is hardly compatible with the low number of frames per second, due to the on-board computation of the SURF descriptors. On-going study is considering more affordable image pre-processings.

Another applicative perspective is to learn a docking controller, enabling robots in a swarm to dock to each other. This goal is amenable to a DIVA approach, as training trajectories can be easily obtained by initially setting the robots in a docked position, and having every robot to undock and start wandering along time. Like for the follower problem, the best state is the initial one and the value of the robot state decreases along time, in the spirit of Murphy's law.

Acknowledgments

The authors gratefully acknowledge the support of the FP7 European Project *Symbrion*, FET IP 216342, `http://symbrion.eu/`, and the ANR Franco-Japanese project Sydinmalas ANR-08-BLAN-0178-01. A part of this research was supported by JST and the grants-in-aid for scientific research on fundamental research (B) 18300047 and on challenging exploratory research 24650070.

REFERENCES

[1] P. Abbeel and A.Y. Ng, 'Apprenticeship learning via inverse reinforcement learning', in *ICML*, ed., Carla E. Brodley, volume 69 of *ACM International Conference Proceeding Series*. ACM, (2004).

[2] Riad Akrour, Marc Schoenauer, and Michèle Sebag, 'Preference-based policy learning', In Gunopulos et al. [12], pp. 12–27.

[3] Riad Akrour, Marc Schoenauer, and Michèle Sebag, 'April: Active preference-based reinforcement learning', in *ECML/PKDD*, p. to appear, (2012).

[4] G. Bakir, T. Hofmann, B. Scholkopf, A.J. Smola, B. Taskar, and S.V.N. Vishwanathan, *Machine Learning with Structured Outputs*, MIT Press, 2006.

[5] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool, 'Surf: Speeded up robust features', in *Computer Vision - ECCV 2006*, volume 3951 of *Lecture Notes in Computer Science*, pp. 404–417, (2006).

[6] Steffen Bickel, Christoph Sawade, and Tobias Scheffer, 'Transfer learning by distribution matching for targeted advertising', in *Advances in Neural Information Processing Systems, NIPS 21*, pp. 145–152. MIT Press, (2008).

[7] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Santa Fe Institute Studies in the Sciences of Complexity, Oxford University Press, 1999.

[8] Lipson H. Bongard J., Zykov V., 'Resilient machines through continuous self-modeling', *Science*, **314**(5802), 1118 – 1121, (2006).

[9] E. Brochu, N. de Freitas, and A. Ghosh, 'Active preference learning with discrete choice data', in *Advances in Neural Information Processing Systems 20*, pp. 409–416, (2008).

[10] S. Calinon, F. Guenter, and A. Billard, 'On Learning, Representing and Generalizing a Task in a Humanoid Robot', *IEEE transactions on systems, man and cybernetics, Part B. Special issue on robot learning by observation, demonstration and imitation*, **37**(2), 286–298, (2007).

[11] Weiwei Cheng, Johannes Fürnkranz, Eyke Hüllermeier, and Sang-Hyeun Park, 'Preference-based policy iteration: Leveraging preference learning for reinforcement learning', In Gunopulos et al. [12], pp. 312–327.

[12] Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba, and Michalis Vazirgiannis, eds. *Proc. European Conf. on Machine Learning and Knowledge Discovery in Databases, ECML PKDD, Part I*. LNCS 6911, Springer Verlag, 2011.

[13] Thorsten Joachims, 'A support vector method for multivariate performance measures', in *ICML*, eds., Luc De Raedt and Stefan Wrobel, pp. 377–384, (2005).

[14] Thorsten Joachims, 'Training linear svms in linear time', in *KDD*, eds., Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, pp. 217–226. ACM, (2006).

[15] J. Zico Kolter, Pieter Abbeel, and Andrew Y. Ng, 'Hierarchical apprenticeship learning with application to quadruped locomotion', in *NIPS*. MIT Press, (2007).

[16] G. Konidaris, S. Kuindersma, A. Barto, and R. Grupen, 'Constructing skill trees for reinforcement learning agents from demonstration trajectories', in *Advances in Neural Information Processing Systems 23*, pp. 1162–1170, (2010).

[17] Michail Lagoudakis and Ronald Parr, 'Least-squares policy iteration', *Journal of Machine Learning Research (JMLR)*, **4**, 1107–1149, (2003).

[18] A.Y. Ng and S. Russell, 'Algorithms for inverse reinforcement learning', in *Proc. of the Seventeenth International Conference on Machine Learning (ICML-00)*, ed., P. Langley, pp. 663–670. Morgan Kaufmann, (2000).

[19] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, 1998.

[20] Csaba Szepesvári, *Algorithms for Reinforcement Learning*, Morgan & Claypool, 2010.

[21] Gerald Tesauro, 'Programming backgammon using self-teaching neural nets', *Artif. Intell.*, **134**(1-2), 181–199, (2002).

[22] Paolo Viappiani and Craig Boutilier, 'Optimal Bayesian recommendation sets and myopically optimal choice query sets', in *NIPS*, pp. 2352–2360, (2010).

Large Scale Co-Regularized Ranking

Evgeni Tsivtsivadze[1] and **Katja Hofmann**[2] and **Tom Heskes**[3]

Abstract. As unlabeled data is usually easy to collect, semi-supervised learning algorithms that can be trained on large amounts of unlabeled and labeled data are becoming increasingly popular for ranking and preference learning problems [6, 23, 8, 21]. However, the computational complexity of the vast majority of these (pairwise) ranking and preference learning methods is super-linear, as optimizing an objective function over all possible pairs of data points is computationally expensive.

This paper builds upon [16] and proposes a novel large scale co-regularized algorithm that can take unlabeled data into account. This algorithm is suitable for learning to rank when large amounts of labeled and unlabeled data are available for training. Most importantly, the complexity of our algorithm does not depend on the size of the dataset. We evaluate the proposed algorithm using several publicly available datasets from the information retrieval (IR) domain, and show that it improves performance over supervised methods. Finally, we discuss possible implications of our algorithm for learning with implicit feedback in an online setting.

1 Introduction and background

Our paper proposes an algorithm that is applicable to large scale learning to rank. Unlike existing approaches the proposed algorithm can take into account unlabeled data, leading to improved ranking performance. Learning to rank algorithms have been successfully applied to various domains such as IR [12], bioinformatics [13], and automated reasoning [11]. One of the bottlenecks associated with ranking tasks is the quadratic dependence on the size of the dataset. That is, most of the (pairwise) methods suffer from the computational burden of optimizing an objective defined over $O(m^2)$ possible pairs for data points, where m is the size of the dataset. Usually, the complexity of ranking algorithms has super-linear dependency on m, except the work of [16] where the use of stochastic gradient descent on pairs results in an extremely efficient training procedure with strong generalization performance. Pairwise learning to rank with stochastic gradient descent results in a scalable methodology that we refer to as stochastic pairwise descent (SPD).

In [16] it is demonstrated that such stochastic gradient based learning methods provide state of the art results using a fraction of a second of CPU time for training. However, SPD is only applicable to supervised learning problems. A natural and useful extension of SPD is an extension to semi-supervised learning, where, in addition to labeled data, a large amount of unlabeled data is used for training. We address this problem and present a large scale co-regularized ranking algorithm for semi-supervised tasks.

Our large scale co-regularized ranking algorithm (LCRA) is formulated within a multi-view framework. In this framework the dataset attributes (i.e., features) are split into independent sets and an algorithm is trained based on these different "views". The goal of the learning process is to find a prediction function for every view performing well on the labeled data in such a way that all prediction functions agree on the unlabeled data. Closely related to this approach is the *co-regularization* framework described in [18], where the same idea of agreement maximization between the predictors is central. Recently it has been demonstrated that the co-regularization approach works well for various tasks e.g. domain adaptation [7], classification, regression [2], and clustering [3]. Moreover, theoretical investigations demonstrate that co-regularization reduces the Rademacher complexity by an amount that depends on the "distance" between the views [15, 19].

We think that our co-regularization algorithm is particularly promising in online learning to rank for IR settings, where a search engine learns from the limited feedback that can be inferred from direct interactions with a search engine users. In this paper, we first focus on co-regularization (in Section 2) and our co-regularized pairwise learning algorithm (3 and 4). Finally, we discuss possible extensions to the online learning to rank for IR setting (Section 5).

2 Large Scale Pairwise Learning to Rank and Co-regularization

Consider a training set $D = (X, Y, Q)$, where $X = (\mathbf{x}_1, \ldots, \mathbf{x}_m)^T \in \mathbb{R}^m$ contains $n-$dimensional feature vectors of the data points, $Y = (y_1, \ldots, y_m)^T \in \mathbb{N}^m$ are the scores/ranks, and $Q = (q_1, \ldots, q_m)^T \in \mathbb{N}^m$ are the indices for identification to which group/query a particular data point belongs. Note that each entry $x_i \in X$ consists of features that encode the relation between a particular item (e.g., a document) to group or query q_i. In addition to the training set $D = (X, Y, Q)$ with *labeled* data we have a training set $\hat{D} = (\hat{X}, \hat{Q})$ with *unlabeled* data points. Also, let us consider M different hypotheses spaces $\mathcal{H}_1, \ldots, \mathcal{H}_M$ or so-called views. These views stem from different representations

[1] Institute for Computing and Information Sciences, Radboud University, The Netherlands & MSB Group, The Netherlands Organization for Applied Scientific Research, Zeist, The Netherlands. Email: evgeni@science.ru.nl

[2] Intelligent Systems Lab Amsterdam, University of Amsterdam, The Netherlands. Email: k.hofmann@uva.nl

[3] Institute for Computing and Information Sciences, Radboud University, The Netherlands. Email: t.heskes@science.ru.nl

of the data points, unique subsets of features. Finally, we define a set of candidate pairs P and $\hat{P}$ (implied by the datasets D and $\hat{D}$) as the set of all tuples $((\mathbf{a}, y_a, q_a), (\mathbf{b}, y_b, q_b))$ and $((\mathbf{a}, q_a), (\mathbf{b}, q_b))$, where $y_a \neq y_b$ and $q_a = q_b$.

Co-regularized algorithms are usually not straightforwardly applicable to large scale learning tasks, when large amounts of unlabeled as well as labeled data are available for training. Several recently proposed algorithms have complexity that is linear in the number of unlabeled data points and super-linear in the number of labeled examples (e.g. cubic as in case of co-regularized least squares [2, 23]). Such methods become infeasible to use as the dataset size increases. In particular, this applies to the pairwise learning setting (note that in the worst case $|P|$ grows quadratically with the dataset size).

2.1 Constructing the pairs

In the pairwise learning setting it is important to be able to sample from $|P|$ and $|\hat{P}|$ without explicitly constructing the datasets of pairs. Several approaches to address this problem are suggested in [16]. For simplicity we adopt one of the most basic techniques: we repeatedly select two examples $(\mathbf{a}, y_a, q_a)$ and $(\mathbf{b}, y_b, q_b)$ from the data until a pair is found such that $y_a \neq y_b$ and $q_a = q_b$. Then we construct a feature vector of the corresponding pair as $\mathbf{p} = \mathbf{a} - \mathbf{b}$ and the label $y = y_a - y_b$.

3 The Algorithm

Stochastic gradient based algorithms are amongst the most popular approaches for large scale learning. Methods such as Pegasos [17], LaSVM [1], GURLS [22] and many others have been successfully applied to large scale classification and regression problems, leading to state-of-the-art generalization performance. Recently, the SPD algorithm [16] has been successfully used to address large scale learning to rank tasks. The main idea is to sample candidate pairs from P for stochastic steps, without constructing P explicitly. This avoids dependence on $|P|$. In essence, the approach proposed in [16] reduces learning to rank to learning a binary classifier via stochastic gradient descent. This reduction preserves the convergence properties of stochastic gradient descent. Our algorithm is related to the above mentioned methods but is preferable in case unlabeled data points are available for learning.

Let us consider the large scale co-regularized ranking algorithm. We write the objective function as

$$J(W) = \sum_{v=1}^{M} \left(\sum_{i=1}^{|P|} \mathcal{L}(\mathbf{p}_i^v, y_i; \mathbf{w}^v) + \lambda \mathcal{L}_R(\mathbf{w}^v) \right) \tag{1}$$

$$+\mu \sum_{\substack{v,u=1 \\ v \neq u}}^{M} \sum_{i=1}^{|\hat{P}|} \mathcal{L}_C(\mathbf{p}_i^v, \mathbf{p}_i^u; \mathbf{w}^v, \mathbf{w}^u),$$

where the first term corresponds to the loss function on the labeled pairs and the second term to a regularization on the individual prediction functions. The third is the co-regularization term that measures the disagreement between the different prediction functions on unlabeled pairs. Note that $W \in \mathbb{R}^{M \times n}$ is a matrix containing weight vectors for different views. Once the model is trained the final prediction can be obtained, for example, by averaging individual predictions for different views (as in [15]). We can approximate the optimal solution (obtained when minimizing (1)) by means of gradient descent

$$\mathbf{w}_{t+1}^v = \mathbf{w}_t^v - \eta_t^v \nabla_{\mathbf{w}^v} J(W). \tag{2}$$

Let us consider the setting in which the squared loss function is used for co-regularization, and the L_2 norm is used for regularization. Choosing the squared loss for the co-regularization term is quite natural as it penalizes the differences among the prediction functions constructed for multiple views (similar to the standard regression setting where the differences between the predicted and true scores are penalized). For every iteration t of the algorithm, we first construct pairs via the procedure described in section 2.1 and denote the set of selected pairs by $A_t \subseteq P$ of size k. Similarly we choose $\hat{A}_t \subseteq \hat{P}$ of size l for each round t on the unlabeled dataset. Let us also denote by A_t^v the set A_t as seen in the view v. Then, we replace the "true" objective (1) with an approximate objective function and write the update rule as follows

$$\mathbf{w}_{t+1}^v = (1 - \eta_t^v \lambda)\mathbf{w}_t^v - \eta_t^v \sum_{(\mathbf{p}, y \in A_t^v)} \nabla \mathcal{L}(\mathbf{p}^v, y; \mathbf{w}_t^v) \; -$$

$$4\mu\eta_t^v \sum_{\substack{v,u=1 \\ v \neq u}}^{M} \sum_{(\mathbf{p}, y \in \hat{A}_t^v \cup \hat{A}_t^u)} \left(\mathbf{w}_t^{vT} \mathbf{p}^v - \mathbf{w}_t^{uT} \mathbf{p}^u \right) \mathbf{p}^v. \tag{3}$$

Note that if we choose A_t to contain a single randomly selected pair, we recover a variant of the stochastic gradient method. In general, we allow A_t to be a set of k and $\hat{A}_t$ to be a set of l data points sampled i.i.d. from P and $\hat{P}$, respectively.

Recall that in the setting described above we are solving a learning to rank problem via reduction to classification of pairs of data points. For classification tasks, the hinge loss is usually considered as more appropriate, although in several studies it has been empirically demonstrated that the squared loss often leads to similar performance (see [14, 27]). The update rule using the hinge loss is derived as follows. Let us define A^{v+} to be the set of examples for which $\mathbf{w}^v$ obtains a non-zero loss, that is $A^{v+} = \{(\mathbf{p}^v, y) \in A_t^v : y\langle \mathbf{p}^v, \mathbf{w}^v \rangle < 1\}$. Then by substituting the second term in equation (3) with $\eta_t^v \sum_{(\mathbf{p}, y \in A^{v+})} y\mathbf{p}^v$ we obtain the update rule for the large scale co-regularized algorithm with hinge loss. When the squared loss function is used for labeled and unlabeled data we obtain the update rule by substituting the second term in equation (3) with $\eta_t^v \sum_{(\mathbf{p}, y \in A_t^v)} (y - \mathbf{w}^{vT}\mathbf{p}^v)\mathbf{p}^v$. Our large scale co-regularized ranking algorithm has complexity of $O(Md)$, where d is the number of nonzero elements in $\mathbf{p}^v$. The pseudocode is shown in Algorithm 1.

4 Experiments

The task of ranking query-document pairs is a problem central to document retrieval - given a query some of the available documents are more relevant in regards to it than some others. Because the user will usually be most interested in the top results returned, document retrieval systems are typically evaluated using performance measures such as mean average precision (MAP).

Algorithm 1 Large scale co-regularized ranking algorithm (LCRA-k-l)

```
Require: Datasets $D$ and $\hat{D}$, regularization parameter $\lambda$, batch sizes $k$ and $l$, number of iterations $N$, number of views $M$, co-regularization parameter $\mu$.
Ensure: $W = 0$
for $t = 1, 2, \dots, N$ do
    Construct $A_t \subseteq P$ (using procedure from Sec 2.1), where $|A_t| = k$ and $\hat{A}_t \subseteq \hat{P}$, where $|\hat{A}_t| = l$
    for $v = 1, 2, \dots, M$ do
        Set $A_t^{v+} = \{(\mathbf{p}^v, y) \in A_t^v : y\langle \mathbf{p}^v, \mathbf{w}_t^v \rangle < 1\}$
        Set $\eta_t^v = \frac{1}{\lambda t}$
        $\mathbf{w}_{t+1}^v \leftarrow (1 - \eta_t^v \lambda)\mathbf{w}_t^v - \eta_t^v \sum_{(\mathbf{p}, y \in A_t^{v+})} y\mathbf{p}^v - 4\mu\eta_t^v \sum_{\substack{v,u=1 \\ v \neq u}}^{M} \sum_{(\mathbf{p}, y \in \hat{A}_t^v \cup \hat{A}_t^u)} \left(\mathbf{w}_t^{vT}\mathbf{p}^v - \mathbf{w}_t^{uT}\mathbf{p}^u\right) \mathbf{p}^v$
Output $W$
```

To benchmark the performance of our algorithm we use Letor 3.0 (LEarning TO Rank) - a collection of several datasets extracted from corresponding IR data collections. The whole collection consists of a set of document-query pairs. Each document-query pair is represented as an example with a quite small number of highly abstract features. Our experiments are performed on each of the datasets separately. We preprocess the datasets by normalizing all feature values to values between 0 and 1 on a per query basis. We use the classical learning setting, where 70% of the data is used for training and the remaining 30% as testing. To simulate a semi-supervised learning setting, a subset of 20% of the training data is randomly selected to be used as labeled data. From the remaining training data, labels are removed.

We compare the performance of our large scale co-regularized ranking algorithm with several other methods, namely the baseline - supervised - version of the algorithm (without co-regularization), which is in equivalent to SPD [16] and to the pairwise Pegasos algorithm [17]. We also compare with the multi-view version of the algorithm, also excluding the co-regularization term, referred to as SPD MV. The comparison is made with several instantiations of the large scale co-regularized ranking algorithm, termed as LCRA-k-l, using various sizes of unlabeled batch examples. For the supervised learning algorithms, only the labeled part of the dataset is used for training. The same set is then used for training the co-regularized model, together with the unlabeled data.

Parameter selection for each model is done by 5-fold cross-validation over the training partition of the data. For the supervised models, parameters to be selected are learning rate η_0 and regularization parameter λ. For the supervised and semi-supervised multi-view models we consider two views that are constructed via random partitioning of the data attributes into two unique sets. Such division of the attributes for constructing multiple views has been previously used in [2]. For the multi-view model we have to estimate the learning rate η_0, as well as the λ_1 and λ_2 parameters. The semi-supervised model has an additional parameter μ controlling the influence of the co-regularization on model selection.

The results of our experiments are included in Table 1. It can be observed that in all experiments the proposed LCRA algorithm outperforms supervised learning methods. The obtained results are expected, as it has been previously demonstrated that co-regularization leads to improved classification performance. Note that our algorithm can be considered a pairwise classification approach for learning to rank.

Dataset	LCRA-1-5	LCRA-1-1	SPD	SPD MV
TD2003	0.15	0.11	0.10	0.11
TD2004	0.14	0.12	0.10	0.10
NP2003	0.54	0.54	0.47	0.49
NP2004	0.51	0.48	0.44	0.45
HP2003	0.60	0.57	0.50	0.52
HP2004	0.52	0.50	0.45	0.45
OHSUMED	0.33	0.31	0.29	0.30

Table 1. MAP-performance comparison of the LCRA algorithm and the baseline methods on the Letor dataset. Note that results of supervised learning algorithms are not comparable to previously reported benchmarks on Letor dataset due to the fact that they are trained only on 20% of the labeled data points.

5 Co-regularization in Online Learning to Rank for IR

In the previous sections we introduced a co-regularization algorithm for semi-supervised learning to rank. We think that this approach is particularly promising in the context of online learning to rank for IR, and discuss its application below.

In online learning for IR a search engine learns improved ranking functions by directly interacting with a user[4] [10, 25]. It is typically modeled as a contextual bandit problem[5] [20], where the query is the context provided by the user, and feedback can be inferred from user clicks on result documents that the search engine returned in response to the query.

5.1 Online learning for IR

The most important difference between the online learning to rank setting and the traditionally considered supervised learning to rank for IR setting is the feedback available to the learning algorithm. Like in other reinforcement learning (RL) settings [20], a retrieval system that learns from user interactions can only infer feedback about the documents or document rankings that it actually presents to the user (and that are inspected by the user). This results in much more limited information to learn from than is available in the supervised

[4] Here we use *online* in the RL sense, meaning that learning and application of the learned solution are performed at the same time. Note that this differs from the term's use in the optimization literature, where is usually refers to the scalability of algorithms.

[5] Contextual bandit problems are a type of reinforcement learning problem actions (i.e., presented documents) depend on the context (i.e., the query), but not on previous interactions between system and environment.

setting, where it is assumed that labels are exhaustive [24], or sampled in some systematic way [4].

In addition to the limited *amount* of feedback available in the online learning to rank for IR setting, the *quality* of the feedback is constrained as follows. Users of a retrieval system expect a ranked list of results, that is more or less ordered by the usefulness of these results given their information need (expressed as e.g., a text query). Consequently, they are most likely to inspect results presented at the top of the returned result list, and continue examining and/or interacting with documents at subsequently lower ranks until their information need is met, or until they run out of time, patience, or some other restricted resource. For the most effective learning outcomes, this means that the documents (or pairs of documents) on which feedback would be most useful for learning should be presented first (we call this strategy of presenting result documents *exploration*). However, the documents that are most useful for learning may not be the ones that are most likely to fulfill the users information need (*exploitation*). Consequently, an effective learning algorithm should balance exploration and exploitation to optimize online performance, i.e., performance while learning from user interactions.

5.2 Related work

Several recent approaches address the problem of learning to rank for IR in an online setting. Yue et al. formulate the Dueling Bandit problem [26] and the K-armed Bandit problem [25]. In both formulations, learning is based on observing pairwise feedback on complete rankings, obtained using so-called interleaved comparison methods, where user clicks are observed on specially constructed result lists that allow inferring a preference between the two rankings [5, 9]. The algorithms proposed to address these tasks follow an exploit-then-explore approach, where it is assumed an algorithm can learn quickly before starting to exploit what has been learned, and performance while learning is largely ignored.

Follow-up work suggests that online performance can be improved by balancing exploration and exploitation [10]. In this work, it was shown that different learning approaches are affected by a balance of exploration and exploitation in different ways. For the listwise Dueling Bandit approach, it was shown that the originally proposed, purely exploratory algorithm over-explored, and that effective learning could be achieved by injecting only two exploratory documents into an otherwise exploitative result list. For the alternative pairwise approach, that is the most similar to the SPD approach evaluated in this paper, it was found that a purely exploitative algorithm performed very well when feedback could be reliably inferred from user clicks. However, when user feedback was noisy, bias introduced by the preference of users for higher-ranked results caused learning outcomes to deteriorate dramatically. To combat this performance loss, exploration had to be increased (which, in its simplest form can consist of random exploration).

5.3 Relation to online co-regularization

The co-regularization algorithm presented in this paper is directly applicable to existing pairwise learning approaches for the online learning to rank for IR setting. Because labeled feedback is particularly limited in the start-up phase of an online learning task, high initial learning gains are expected in this setting when easily obtainable unlabeled data can be used to complement this data. The resulting higher-quality result lists are expected to result in more reliable feedback [10]. As a result, learning could be sped up while reducing the need for exploration, leading to increased online performance. An experimental investigation of this hypothesis will be conducted in follow-up work.

6 Conclusion

In this paper we have presented a large-scale co-regularized algorithm for ranking and preference learning. Our algorithm can use unlabeled data to improve learning when labeled data is scarce. Our experiments on 7 standard learning to rank data sets show that our co-regularization component consistently improves performance over algorithms without co-regularization. We think that this approach can be particularly beneficial in an online learning to rank setting, where algorithms learn directly from interacting with users and effective use of limited feedback is paramount. We conclude with a brief outlook on future work in this area.

Acknowledgments

This research was partially supported by the Netherlands Organisation for Scientific Research (NWO) under project nr 612.061.814 and a Vici grant 639.023.604.

References

[1] Leon Bottou, Antoine Bordes, and Seyda Ertekin. Lasvm, 2009. `http://mloss.org/software/view/23/`.

[2] Ulf Brefeld, Thomas Gärtner, Tobias Scheffer, and Stefan Wrobel, 'Efficient co-regularised least squares regression', in *ICML '06*, pp. 137–144. ACM, (2006).

[3] Ulf Brefeld and Tobias Scheffer, 'Co-em support vector learning', in *ICML'04*, p. 16, New York, NY, USA, (2004). ACM.

[4] Ben Carterette, James Allan, and Ramesh Sitaraman, 'Minimal test collections for retrieval evaluation', in *SIGIR '06*, pp. 268–275. ACM, (2006).

[5] Olivier Chapelle, Thorsten Joachims, Filip Radlinski, and Yisong Yue, 'Large-scale validation and analysis of interleaved search evaluation', *ACM Trans. Inf. Syst.*, **30**(1), 6:1–6:41, (2012).

[6] W Chu and Z Ghahramani, 'Extensions of Gaussian processes for ranking: semi-supervised and active learning', in *NIPS Workshop on Learning to Rank*, pp. 29–34, (2005).

[7] Hal Daume, Abhishek Kumar, and Avishek Saha, 'Co-regularization based semi-supervised domain adaptation', in *NIPS '10*, eds., J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, 478–486, (2010).

[8] Kevin Duh and Katrin Kirchhoff, 'Semi-supervised ranking for document retrieval', *Comput. Speech Lang.*, **25**(2), 261–281, (2011).

[9] K. Hofmann, S. Whiteson, and M. de Rijke, 'A probabilistic method for inferring preferences from clicks', in *CIKM '11*, pp. 249–258, (2011).

[10] K. Hofmann, S. Whiteson, and M. de Rijke, 'Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval', *Information Retrieval Journal (to appear)*, (2012).

[11] Daniel Kühlwein, Josef Urban, Evgeni Tsivtsivadze, Herman Geuvers, and Tom Heskes, 'Multi-output ranking for automated reasoning', in *KDIR'11*, (2011).

[12] Tie-Yan Liu, 'Learning to rank for information retrieval', *Foundations and Trends in Information Retrieval*, **3**(3), 225–331, (2009).

[13] Iain Melvin, Jason Weston, Christina S. Leslie, and William Stafford Noble, 'Rankprop: a web server for protein remote homology detection', *Bioinformatics*, **25**(1), 121–122, (2009).

[14] Ryan Rifkin, Gene Yeo, and Tomaso Poggio, 'Regularized least-squares classification', in *Advances in Learning Theory: Methods, Model and Applications*, eds., J.A.K. Suykens, G. Horvath, S. Basu, C. Micchelli, and J. Vandewalle, volume 190 of *NATO Science Series III: Computer and System Sciences*, chapter 7, 131–154, IOS Press, Amsterdam, (2003).

[15] David Rosenberg and Peter L. Bartlett, 'The Rademacher complexity of co-regularized kernel classes', in *AISTATS'07*, eds., Marina Meila and Xiaotong Shen, pp. 396–403, (2007).

[16] D. Sculley, 'Large Scale Learning to Rank', in *NIPS Workshop on Advances in Ranking*, (2009).

[17] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro, 'Pegasos: Primal Estimated sub-GrAdient SOlver for SVM', in *ICML '07*, pp. 807–814. ACM, (2007).

[18] Vikas Sindhwani, Partha Niyogi, and Mikhail Belkin, 'A co-regularization approach to semi-supervised learning with multiple views', in *ICML Workshop on Learning with Multiple Views*, (2005).

[19] Vikas Sindhwani and David Rosenberg, 'An RKHS for multi-view learning and manifold co-regularization', in *ICML'08*, eds., Andrew McCallum and Sam Roweis, pp. 976–983, Helsinki, Finland, (2008).

[20] Richard S. Sutton and Andrew G. Barto, *Reinforcement learning: An introduction*, MIT Press, Cambridge, MA, USA, 1998.

[21] Martin Szummer and Emine Yilmaz, 'Semi-supervised learning to rank with preference regularization', in *CIKM '11*, pp. 269–278. ACM, (2011).

[22] Andrea Tacchetti, Pavan Mallapragada, Matteo Santoro, and Lorenzo Rosasco, 'GURLS: a toolbox for large scale multiclass learning', in *NIPS 2011 workshop on parallel and large-scale machine learning*. `http://cbcl.mit.edu/gurls/`.

[23] Evgeni Tsivtsivadze, Tapio Pahikkala, Jorma Boberg, Tapio Salakoski, and Tom Heskes, 'Co-regularized least-squares for label ranking', in *Preference Learning*, eds., Eyke Hüllermeier and Johannes Fürnkranz, pp. 107–123, (2010).

[24] Ellen M. Voorhees and Donna K. Harman, *TREC: Experiment and Evaluation in Information Retrieval*, Digital Libraries and Electronic Publishing, MIT Press, 2005.

[25] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims, 'The k-armed dueling bandits problem', *Journal of Computer and System Sciences*, **78**(5), 1538 – 1556, (2012).

[26] Yisong Yue and Thorsten Joachims, 'Interactively optimizing information retrieval systems as a dueling bandits problem', in *ICML'09*, pp. 1201–1208, (2009).

[27] Peng Zhang and Jing Peng, 'Svm vs regularized least squares classification', in *Proceedings of the International Conference on Pattern Recognition*, ICPR '04, pp. 176–179, (2004).

First Steps Towards Learning from Game Annotations

Christian Wirth and **Johannes Fürnkranz** [1]

Abstract.
Most of the research in the area of evaluation function learning is focused on self-play. However in many domains, like chess, expert feedback is amply available in the form of annotated games. This feedback comes usually in the form of qualitative information due to the inability of humans to determine precise utility values for game states. We are presenting a first step towards integrating this qualitative feedback into evaluation function learning by reformulating it in terms of preferences. We extract preferences from large-scale database for annotated chess games and use them for calculating the feature weights of a heuristic chess position evaluation function. This is achieved by extracting the feature weights out of the linear kernel from a learned SVMRANK model, based upon the given preference relations. We evaluate the resulting function by creating multiple heuristics based upon different sized subsets of the trainings data and compare them in a tournament scenario. Although our results did not yield a better chess playing program, the results confirm that preferences derived from game annotations may be used to learn chess evaluation functions.

1 Introduction

For many problems, human experts are able to demonstrate good judgment about the quality of certain courses of actions or solution attempts. Typically, this information is of qualitative nature (e.g., “A treatment a is more effective than treatment b), and cannot be expressed numerically without selecting arbitrary values. This is due to the fact that humans are not able to determine a precise utility value of an option, but are typically able to compare the quality of two options. The emerging field of preference learning tries to make this information usable in the field of machine learning, by introducing concepts and methods for applying qualitative preferences to a wide variety of learning problems [12].

In the game of chess, qualitative human feedback is amply available in the form of game notations. For example, the company *Chessbase*[3] specializes in the collection and distribution of chess databases. Their largest database contains annotations for over 66000 games, according to the official page. However, this rich source of information about the game has so far been ignored in the literature on machine learning in chess [21, 11]. Much of the work in this area has concentrated on the application of reinforcement learning algorithms to learn meaningful evaluation functions [3, 4, 7]. These approaches have all been modeled after the success of *TD-Gammon* [24], a learning system that uses temporal-difference learning [22] for training a game evaluation function [23]. However, all these algorithms were trained exclusively on self-play, entirely ignoring human feedback that is readily available in annotated game databases.

In this paper, we report the results of a first study that aims at learning a heuristic function for chess based on a large amount of qualitative feedback from experts available in annotated game database. In particular, we show how preferences can be extracted from chess databases, and show how state-of-the-art ranking algorithms can be used to successfully learn an evaluation function. The learning setup is based on the methodology used in [16], where it has been used for learning evaluation functions from move preferences of chess players of different strengths. However, to our knowledge this is the first work that reports on results from learning evaluation functions from game annotations.

In Section 2, we are explaining which information can be contained in annotations for chess games, especially concerning *portable game notation* files with *numeric annotation glyphes* [8]. A widely available data format. Section 3 details the object ranking by preferences method in general, as well as how to extract the preference information. In our experimental setup (Section 5), we are training a SVM with the preference data, based upon the state feature values given by a strong chess engine. This enables the creation of a new heuristic evaluation function by using the learned (linear) SVM model. The quality of the resulting function is evaluated in a chess engine tournament. Section 7 is concluding the paper and gives a short overview over possible further work.

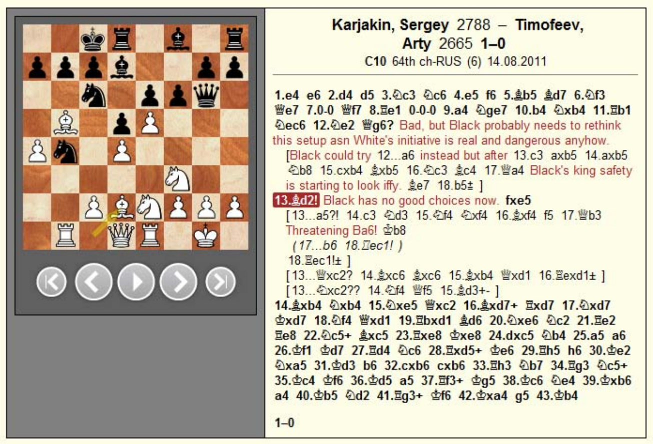
2 Game Annotations in Chess

Chess is a game of great interest, which has generated a large amount of literature that analyzes the game. Particularly popular are game annotations, which are frequently published after important or interesting games have been played in tournaments. These annotations reflect the analysis of a particular game by a (typically) strong professional chess player., They have been produced without any time constraints, and the annotators can resort to any means they deem necessary for improving their judgement (such as consulting colleagues, books, or computers). Thus, these annotations are usually of a high quality.

Annotated chess games are amply available, not only in chess books or magazines. Chess databases, such as those provided by companies like *Chessbase*[3], are storing millions of games, many of them annotated. Chess players of all strengths use them regularly to study the game or to prepare against their next opponent.

Chess annotators use a standardized set of symbols for annotating moves and positions, which have been popularized by the Chess Informant book series. *Portable game notation* (PGN) files are chess games recorded in *standard algebraic notation* with optional *numeric annotation glyphes* (NAG) [8]. Those annotation symbols can be divided into three major categories: *move*, *position* and *time* evaluation.

[1] TU Darmstadt, Germany, [cwirth, juffi]@ke.tu-darmstadt.de



Karjakin, Sergey 2788 – **Timofeev, Arty** 2665 **1–0**
C10 64th ch-RUS (6) 14.08.2011

1.e4 e6 2.d4 d5 3.♘c3 ♘c6 4.e5 f6 5.♗b5 ♗d7 6.♘f3 ♕e7 7.0-0 ♕f7 8.♖e1 0-0-0 9.a4 ♘ge7 10.b4 ♘xb4 11.♖b1 ♘ec6 12.♘e2 ♕g6? Bad, but Black probably needs to rethink this setup asn White's initiative is real and dangerous anyhow.
[Black could try 12...a6 instead but after 13.c3 axb5 14.axb5 ♘b8 15.cxb4 ♗xb5 16.♘c3 ♗c4 17.♕a4 Black's king safety is starting to look iffy. ♗e7 18.b5⩲]
13.♗d2! Black has no good choices now. **fxe5**
[13...a5?! 14.c3 ♘d3 15.♘f4 ♘xf4 16.♗xf4 f5 17.♕b3 Threatening Ba6! ♔b8
(17...b6 18.♖ec1!)
18.♖ec1!⩲]
[13...♕xc2? 14.♗xc6 ♗xc6 15.♗xb4 ♕xd1 16.♖exd1±]
[13...♘xc2?? 14.♘f4 ♕f5 15.♗d3+-]
14.♗xb4 ♘xb4 15.♘xe5 ♕xc2 16.♗xd7+ ♖xd7 17.♘xd7 ♔xd7 18.♘f4 ♕xd1 19.♖bxd1 ♗d6 20.♘xe6 ♘c2 21.♖e2 ♖e8 22.♘c5+ ♗xc5 23.♖xe8 ♔xe8 24.dxc5 ♘b4 25.a5 a6 26.♔f1 ♔d7 27.♖d4 ♘c6 28.♖xd5+ ♔e6 29.♖h5 h6 30.♔e2 ♘xa5 31.♔d3 b6 32.cxb6 cxb6 33.♖h3 ♘b7 34.♖g3 ♘c5+ 35.♔c4 ♔f6 36.♔d5 a5 37.♖f3+ ♔g5 38.♔c6 ♘e4 39.♔xb6 a4 40.♔b5 ♘d2 41.♖g3+ ♔f6 42.♔xa4 g5 43.♔b4

1–0

Figure 1. An annotated chess game (screen-shot taken from `http://chessbase.com/`).

move evaluation: Each move can be annotated with a symbol indicating its quality. Six symbols are commonly used:

- very poor move (??),
- poor move (?),
- speculative move (?!),
- interesting move (!?),
- good move (!),
- very good move (!!).

position evaluation: Each move can be annotated with a symbol indicating the quality of the position it is leading to:

- white has a decisive advantage (+−),
- white has a moderate advantage (±),
- white has a slight advantage (⩲),
- equal chances for both sides (=),
- black has a slight advantage (⩱),
- black has a moderate advantage (∓),
- black has a decisive advantage (−+),
- the evaluation is unclear (∞).

time evaluation: Each move can be annotated with a symbol indicating a time constraint that arose at this move. This information is not used in our experiments.

In addition to annotating games with NAG symbols, annotators can also add textual comments and move variations to the game, i.e., in addition to the moves that have actually been played in the course of the game, an annotator provides alternative lines of play. Those are usually suggestions in the form of short move chains that are leading to more promising states than the move chain used in the real game. Variations can also have NAG symbols, and may contain subvariations.

Figure 1 shows an example for an annotated chessgame. The left-hand side shows the game position after the 13th move of white. Here, black is in a difficult position after the mistake he made. (12...♕g6?). From the suggested moves, 13...a5?! is the best, but even here white has the upper hand at the end of the variation (18.♖ec1!⩲), as well as in the end of the suggested move chain starting with 13...♕×c2 . On the other hand, 13...♘×c2?? is an even worse choice, ending in a position that is clearly lost for black (+−).

It is important to note that this feedback is of qualitative nature, i.e., it is not clear what the expected reward is in terms of, e.g., percentage of won games from a position with evaluation ±. However, it is clear that positions with evaluation ± are preferable to positions with evaluation ⩲ or worse (=, ⩱, ∓, −+).

Also note that the feedback for positions typically applies to the entire sequence of moves that has been played up to reaching this position (a *trajectory* in reinforcement learning terminology). The qualitative position evaluations may be viewed as providing an evaluation of the trajectory that lead to this particular position, whereas the qualitative move evaluations may be viewed as evaluations of the expected value of a trajectory that starts at this point.

However, even though there is a certain correlation between these two types of annotations (good moves tend to lead to better positions and bad moves tend to lead to worse positions), they are not interchangable. A very good move may be the only move that saves the player from imminent doom, but must not necessarily lead to a very good position. Conversely, a bad move may be a move that misses a chance to mate the opponent right away, but the resulting position may still be good for the player.

3 Learning an Evaluation Function from Preferences

For learning the mentioned SVM model, it is required to formulate the task as a binary classification problem. We are showing how this can be done by using preference learning.

3.1 Preference Learning

Preference learning is about inducing predictive preference models from empirical data. This establishes a connection between machine learning and research fields like preference modeling or decision making. Especially "learning to rank by preferences" is deemed promising by the community. Preference learning can be applied to *label ranking*, by defining preferences over a set of labels concerning a specific set of objects [25]. But it is also possible to define preference directly over a set of objects, for creating a ranking of those objects [14]. Preferences themselves are constraints that can be violated, which leads to higher flexibility concerning the solving process, opposed to hard constraints. These constraints can be described via a *utility function* or *preference relations*. [12] We are only considering preference relations in this work, because they can be represented in a qualitative manner.

Object Ranking is about learning how to order a subset of objects out of a (potentially infinite) reference set $\mathcal{Z}$. Those objects $\mathbf{z} \in \mathcal{Z}$ are usually given as a vector of attribute/value pairs, but this is no necessary property. The trainings data is given in the form of rankings, which is decomposed into a finite set of pairwise preferences $\mathbf{z_i} \succ \mathbf{z_j}$. The object ranker is then learning a ranking function $f(\cdot)$ which returns a (ranked) permutation of a given object set. [12]

3.2 States and Actions

In chess, we are searching for the best action $\boldsymbol{a} \in \mathbf{A}$ for a state $\boldsymbol{s} \in \mathbf{S}$. For game tree exploration concerns or suboptimal play, it can also be required to determine the expected quality of an suboptimal action

$\boldsymbol{a'} \in \mathbf{A}$. When defining this quality in a relative way, as opposed to an absolute value, we are searching for a rank. Because of the high amount of legal states in chess (roughly 10^{50} states [2]), it is not feasible to learn those ranking functions directly. Considering the chess transition function $f : \boldsymbol{S} \times \boldsymbol{A_s} \rightarrow \boldsymbol{S_s}$, with $\mathbf{S_s} \subset \mathbf{S}$ as the set of states that can be reached from $\mathbf{s}$ by an action $\boldsymbol{a} \in \mathbf{A_s}$ possible in $\boldsymbol{s}$, we can rewrite the problem as the search for a ranking for all $\boldsymbol{s} \in \mathbf{S_s}$. This ranking is also not dependent on the current state $\mathbf{s}$, because the state/action history is not relevant for a chess state (excluding the *fifty-moves* and the *threefold repetition* draw rules). This reduces the problem to a *object ranking* problem over all $\boldsymbol{s} \in \mathbf{S}$.

3.3 SVM-based ranking

Following [16], we can use state preferences of the form $\boldsymbol{s}_i \succ \boldsymbol{s}_j$ for training the SVMRank ranking support vector machine proposed by [13].[2] Its key idea is to reinterpret the preference statements as constraints on the evaluation function, i.e.,

$$\boldsymbol{s}_i \succ \boldsymbol{s}_j \Leftrightarrow h(\boldsymbol{s}_i) > h(\boldsymbol{s}_j).$$

If the function h is a linear, i.e., it is a weighted sum

$$h(\boldsymbol{s}) = \sum_f w_f \cdot f(\boldsymbol{s})$$

of features f, the latter part is equivalent to

$$\begin{aligned} h(\boldsymbol{s}_i - \boldsymbol{s}_j) &= \sum_f w_f \cdot f(\boldsymbol{s}_i - \boldsymbol{s}_j) \\ &= \sum_f w_f \cdot (f(\boldsymbol{s}_i) - f(\boldsymbol{s}_j)) > 0 \end{aligned}$$

Thus, essentially, the training of the ranking SVM corresponds to the training of a classification SVM on the pairwise differences $\boldsymbol{s}_i - \boldsymbol{s}_j$ between positions $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$. The pairwise ranking information can thus be converted to binary training data in the form of a feature distance vector $\overrightarrow{A}$ with the preference relation $r \in \{<, >\}$ as the binary class vector.

4 Generating Preference Data from Game Annotations

The training data that are needed for an object ranking algorithm like SVMRank can be generated from game annotations of the type discussed in Section 2. For our first experiments, we only focused on move preferences, and ignored state preferences.

Our algorithm for generating preferences from move annotations is sketched in Algorithm 1: a given list of games $\boldsymbol{G}$ in PGN format is parsed, and triplets $(\boldsymbol{s}, \boldsymbol{a}, \mathbf{n}), \mathbf{n} \in \mathbf{N}_{\boldsymbol{s},\boldsymbol{a}}$ with $\mathbf{N}_{\boldsymbol{s},\boldsymbol{a}}$ being the list of NAG in $(\boldsymbol{s}, \boldsymbol{a})$ are created for each occurrences of a NAG symbol. A state is represented by its Forsyth-Edwards Notation (FEN). It is a serialized representation of the game board, capturing all data that is required to uniquely identify a chess state [8]. Actions are saved in the Long Algebraic Notation (LAN). After collecting this data for every game, all triples containing the same FEN state are compared. The NAG symbols are checked against a static relation list and a pairwise preference relation for the attached actions is created, if possible. The static relation table contains entries like $\mathbf{n_1}$:?? $\mathbf{n_2}$:!? $\rightarrow \mathbf{n_1} < \mathbf{n_2}$. In rare cases, we may get multiple conflicting annotations for a pair $(\boldsymbol{s}, \boldsymbol{a})$, which are then ignored.

Algorithm 1 Preference Generation

```
Require: list of games $G$, initial position $s_0$
 1: $triples \leftarrow \emptyset, prefs \leftarrow \emptyset, seen \leftarrow \emptyset$
 2: for all $g \in G$ do
 3:     $s \leftarrow s_0$
 4:     for all $(a, N_{s,a}) \in g$ do
 5:         $s \leftarrow$ MOVE$(s, a)$
 6:         for all $n \in N_{s,a}$ do
 7:             $triples \leftarrow triples \cup \{(s, a, n)\}$
 8:             $seen \leftarrow seen \cup \{s\}$
 9:         end for
10:     end for
11: end for
12: for all $s' \in seen$ do
13:     for all $N_{\boldsymbol{s'},\boldsymbol{a}}, N_{\boldsymbol{s'},\boldsymbol{a'}} \in triples$ with $\boldsymbol{a} \neq \boldsymbol{a'}$ do
14:         $r \leftarrow$ RELATION$(N_{\boldsymbol{s'},\boldsymbol{a}}, N_{\boldsymbol{s'},\boldsymbol{a'}})$
15:         if $r \neq \varnothing$ then
16:             $s_1 \leftarrow$ MOVE$(s', a)$, $s_2 \leftarrow$ MOVE$(s', a')$
17:             $prefs \leftarrow prefs \cup \{(s_1, s_2, r)\}$
18:         end if
19:     end for
20: end for
21: return $prefs$
```

When applying this algorithm to the example given in Figure 1, it would yield the following action preferences: ($\boldsymbol{s}$, ♕×c2 $\succ$ ♘×c2),($\boldsymbol{s}$,a5 $\succ$ ♘×c2),($\boldsymbol{s}$,a5 $\succ$ ♕×c2) with $\boldsymbol{s}$ being the state shown in the example.

In a last step, action preferences $(\boldsymbol{s}, \boldsymbol{a}_1 \succ \boldsymbol{a}_2)$ are converted to state preferences by applying $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ to $\boldsymbol{s}$, resulting in state preferences $\boldsymbol{s}_1 \succ \boldsymbol{s}_2$, where $\boldsymbol{s}_i = \text{MOVE}(\boldsymbol{s}, \boldsymbol{a}_i)$. A practical problem is, that annotated moves are usually not leading to a stable state to which the qualitative evaluation can be directly applied. For example, in the middle of an exchange sequence, the first player will be behind by one piece after the initial move but may gain a significant advantage after a short chain of moves. For this reason, preferences are not applied to the positions $\boldsymbol{s}_i$, but to quiet positions that result from a fixed-depth search starting in $\boldsymbol{s}_i$ (we use depth 7), followed by a quiescence search. The positions $\bar{\boldsymbol{s}}_i$ at the leaves of these searches are then used in the state preferences.

Additionally, most variations added to the PGN data are also move chains and not single moves, hence we are applying the suggested move chain to the state and not only the first, single move. This is implemented in step 16 of algorithm 1.

5 Experimental Setup

For showing the usefulness of preference data, we are training a SVM model based on preference data generated from annotated chess games (Section 5.1), and employ it in the strong open source chess engine CUCKOO (Section 5.2). All states are represented by the heuristic features created by the position evaluation function. Training a linear kernel model allows us to simply extract the feature weights for the linear sum function. The quality of the preferences can now be analyzed by comparing the playing strength of our reweighted chess engine.

[2] Available from `http://svmlight.joachims.org`.

Feature Type	# Features	Description
material difference	1	Difference in the sum of all piece values per player.
piece square	6	Position dependent piece values by static piece/square tables. A single value for every piece type.
pawn bonus	1	Bonus for pawns that have passed the enemy pawn line , while also considering its distance to the enemy king.
trade bonus	2	Bonus for following the “when ahead trade pieces, when behind trade pawns” rules.
castle bonus	1	Evaluates the castling possibility.
rook bonus	1	Bonus for rooks on (half-) open files.
bishops scores	2	Evaluating the bishops position by attack possibilities, if trapped and relative positioning.
threat bonus	1	Difference in the sum of all piece values under attack.
king safety	1	Evaluates the kings position relative to the rooks.

Table 1. Features used in the linear evaluation function of the Cuckoo chess engine.

5.1 ChessBase

As a data source we are using the *Mega Database 2012*, provided by *Chessbase*.[3] To the authors’ knowledge, it is the largest database of professionally annotated chess games available. The annotations are commonly, but not exclusively provided by chess grandmasters. In this first study, we only considered action preferences, and ignored state preferences, mostly because of complexity considerations.

In the more than 5 million games contained in the database, we identified 86,767 annotated games with 1.67 million annotated moves in total. 343,634 NAG symbols occurred pairwise concerning the same state, but different moves. Out of these, the preference generation process yielded 271,925 preferences with 190,143 being unambiguous and not equal. The rest are incomparable symbol pairs, and were ignored in our data generation process.

5.2 Cuckoo Chess Engine

We used the Cuckoo chess engine[4] for our experiments, because of its combination of high playing strength[5] and good modifiability. It facilitates BitBoards [19, 1] as state representation and NegaScout [18] as search algorithm.

Most state of the art chess engines are using a heuristic position evaluation function, while searching for the best, currently reachable position with enhanced Alpha-Beta search algorithms like NegaScout. For performance reasons, evaluation functions are commonly linear sums over abstract, manually constructed features. Usually, features like material difference or usefulness of pieces in their current position are used. Table 1 shows the 16 features shown by Cuckoo. We used these features for describing a state.

The Cuckoo Chess Engine was used in a single thread configuration. All experiments haven been executed on systems with 2 cores or more, ensuring independence of the available computing power for each player.

5.3 Training Data

In our experimental setup, we are creating the object preferences as described in Section 4. The pairwise preference data is used as training data for SVMRank, which is an optimized implementation of the SVM based ranker described in 2.4, which can handle pairwise preference data directly [13]. The feature weights can now be extracted out of the SVMRank model and be applied to the Cuckoo chess engine.

The features have not been standardized or normalized, because they are already internally normalized to a pico-pawn scale, hence no significant improvement in classification accuracy was expected. This was also confirmed in experiments.

Annotators can disagree concerning the exact quality of a move, but the same relative outcome is expected when comparing two moves. E.g. an annotator may use $\mathbf{n_1}$:? instead of $\mathbf{n_1}$:??, but not $\mathbf{n_1}$:!! if the consensus is $\mathbf{n_1} < \mathbf{n_2}$, $\mathbf{n_2}$:!?. Tests confirmed the expected low amount of directly contradicting preferences ($< 0.2\%$), but it is still possible for subsets to indicate a different valuation of features.

We created 6 different engines, based upon different training set sizes. 5%, 10%, 25%, 50%, 75% and 100% randomly sampled elements of the available preference data have been used to create the different engines. The results have been generated by averaging over three all-against-all tourneys, including the player with the original feature weighting as upper bound and a random player as baseline. The random engine is picking new random weights for each position evaluation. The distribution for those weights is a uniform distribution, bounded by the min/max values observed within all learned SVMRank models. Each pairing played 100 games with a 5min timeframe and no increments.

5.4 Evaluation

All results are reported in terms of Elo ratings [9], which is the commonly used rating system for rating chess players. It not only considers the absolute percentage of won games, but also takes the strength of the opponent into account. A rating difference of a 100 points approximately means that the stronger player has an expected win rate of $5/8$. It also enables the reporting of upper and lower bounds for the playing strength of each player. For calculating the Elo values, a base Elo of 2600 was used, because this it the rating for the Cuckoo Chess Engine as reported by the *Computer Chess Rating List*[6]. It should be noted that computer engine Elo ratings are not directly comparable to human Elo ratings, because they are typically estimated on independent player pools, and thus only reflect relative strengths.

[3] http://www.chessbase.com/

[4] http://web.comhem.se/petero2home/javachess/

[5] http://www.computerchess.org.uk/ccrl/

[6] http://www.computerchess.org.uk/ccrl/

6 Results

6.1 Predictive Accuracy

We first compared the predictive accuracy of different classifiers on the binary classification problem of learning a preference relation from the collected preference set. The binary classification accuracy a can be compared to the average amount of swapped pairs over all pairs metric e of the original ranking problem by $a = 1 - e$. The *Weka*[7] implementation of all classifiers was used, if not stated otherwise.

Table 2 shows that multilayer perceptrons and random forests yielded the best results, whereas LibLinear and SVMRank performed the worst. This seems to indicate that a non-linear combination of the base features is able to yield a better performance than the linear combination that is used in the chess program.

We can also see the performance of the original position evaluation function of Cuckoo, which is a linear function that assigns a uniform weight to all features. IT is somewhat higher than the trained linear functions, but considerably below the best non-linear functions.

Classifier	Accuracy
Multilayer Perceptron [5]	0.6871
Random Forest [6]	0.6864
Naive Bayes Tree [15]	0.6799
J48 [17]	0.6719
Pegasos [20]	0.6651
LIBLinear[8][10]	0.6521
SVMRank[9][13]	0.6505
Cuckoo	0.6620

Table 2. Comparison of the predictive performance of different classifiers and the Cuckoo chess engine (10-fold CV).

6.2 Playing Strength

For evaluating the playing strength we were limited to using a linear evaluation function because only those could be easily plugged into the chess program. We chose evaluation functions learned by SVMRank. Figure 2 shows the development of the rating over the percentage of used preferences in the training data. It is clearly recognizable in that an increase in the amount of used preference data is leading to an improved chess engine, which we take as evidence that the game annotations provide useful information for learning an evaluation function. The playing strength is clearly above the random baseline, which reached an average Elo rating of 2332 ± 32, but well below the original player and its average Elo rating of 2966 ± 43.

6.3 Stability

The player that was trained on 5% of the data is a clear outlier, resulting from the comparably high variance in the training data at this point. The variance of the feature weights at this setting is shown in Figure 3.

However, most features are showing convergence and a mostly stable average value. Figure 4 shows the development of the feature values (average, standard deviation and min/max values) for all features. The 10 features in the left and the middle graph are quite stable, whereas the features in the right graph are rather unstable. For the features *castleBonus* and *bishopB*, a possible explanation could be the sparsity of these values. The feature value difference for these values is 0 in 84.6% and 99.7%, respectively, of all training examples.

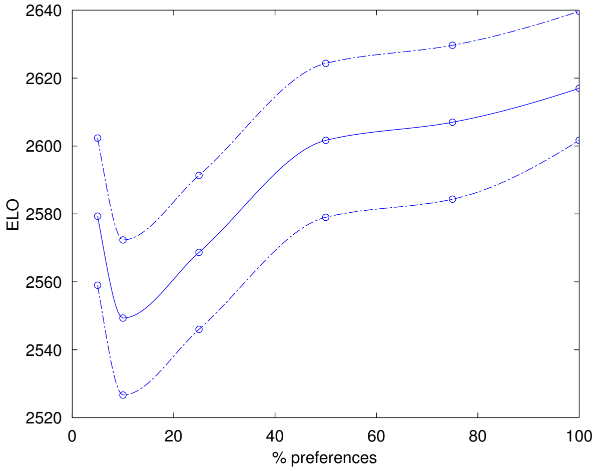


Figure 2. Learning curve, measured in Elo rating.

7 Conclusion

This paper presented the results of a preliminary study that uses expert feedback in the form of game annotations for the automated construction of an evaluation function for the game of chess. It was shown how annotated chess games can be used for the creation of preference data. This is especially interesting because of the

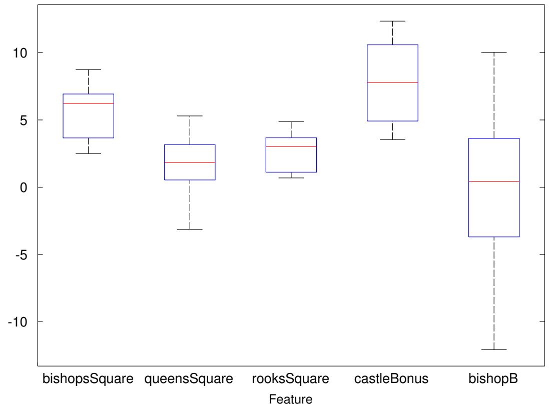


Figure 3. Learned weight for the 5 most variant features, based on 10 different 5% samplings

[7] http://www.cs.waikato.ac.nz/ml/weka/

[8] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

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

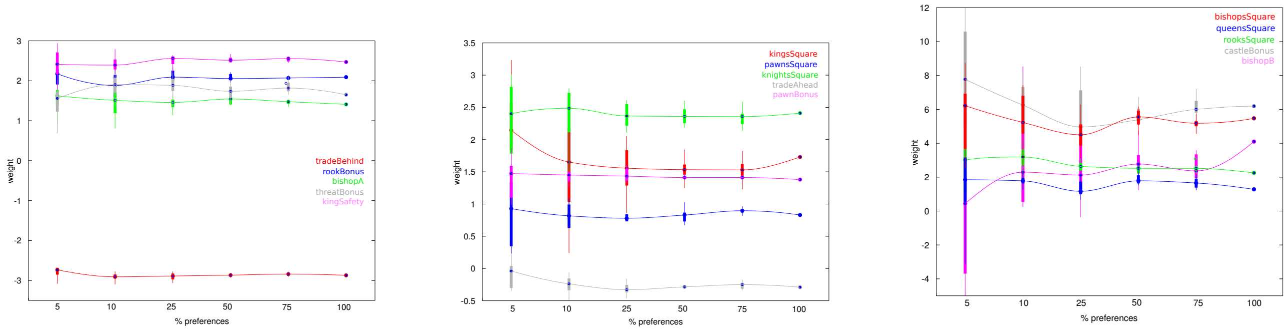


Figure 4. Average and variance for the feature weights, averaged over 10 samples per subset size. Weights are scaled to *materialDifference*= 1.

widespread availability of annotated chess games, which enables the creation of large-scale datasets.

Following the approach shown by Paulsen et al [16], the preferences have been successfully used to learn the feature weights for an position evaluation function. It can be observed, that the playing strength of the chess engine is scaling with the amount of trainings data. This is a first step towards using qualitative feedback in game playing scenarios.

However, alhthough we can observe a correlation with the amount of seen preferences and the playing strength of the learned players, their overall strength was not able to reach the strength of the original player. We still have to investigate the reasons for this, but it should be noted that the original feature weighting is outperforming the learned weights. Thus, SVMRANK was only able to find suboptimal feature weights.

Moreover, in this work we have essentially ignored state preferences and focused on action preferences. The reason for this was pragmatic, because action preferences relate to a single state, whereas state preferences can be widely compared, even across multiple games. For example, every position evaluated with $+-$ can be considered to be better than every position evaluated with $=$, all of which can, in turn, be considered to be preferred over positions that are evaluated with $-+$. This approach gives rise to a vast number of preferences. One could consider to only apply this to positions of the same game, because different annotators may have a different calibration of the used symbols. This would also reduce the complexity. These issues are currently under investigation.

REFERENCES

[1] G. M. Adel'son-Vel'skii, V. L. Arlazarov, A. R. Bitman, A. A. Zhivotovskii, and A. V. Uskov, 'Programming a computer to play chess', *Russian Mathematical Surveys*, **25**(2), 221, (1970).

[2] V. Allis, *Searching for Solutions in Games and Artificial Intelligence*, Ph.D. dissertation, University of Limburg, The Netherlands, 1994.

[3] J. Baxter, A. Tridgell, and L. Weaver, 'Learning to play chess using temporal differences', *Machine Learning*, **40**(3), 243–263, (September 2000).

[4] D. F. Beal and M. C. Smith, 'Temporal difference learning applied to game playing and the results of application to Shogi', *Theoretical Computer Science*, **252**(1-2), 105–119, (2001). Special Issue on Papers from the Computers and Games 1998 Conference.

[5] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford, UK, 1995.

[6] L. Breiman, 'Random forests', *Machine Learning*, **45**(1), 5–32, (2001).

[7] S. Droste and J. Fürnkranz, 'Learning the piece values for three chess variants', *International Computer Games Association Journal*, **31**(4), 209–233, (2008).

[8] S. J. Edwards. Portable game notation, 1994. accessed on 14.06.2012.

[9] A. E. Elo, *The Rating of Chessplayers, Past and Present*, Arco, New York, 2nd edn., 1978.

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, 'Liblinear: A library for large linear classification', *Journal of Machine Learning Research*, **9**, 1871–1874, (2008).

[11] J. Fürnkranz, 'Machine learning in computer chess: The next generation', *International Computer Chess Association Journal*, **19**(3), 147–161, (1996).

[12] J. Fürnkranz and E. Hüllermeier (eds.), *Preference Learning*, Springer-Verlag, 2010.

[13] T. Joachims, 'Optimizing search engines using clickthrough data', in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pp. 133–142. ACM Press, (2002).

[14] T. Kamishima, H. Kazawa, and S. Akaho, 'A survey and empirical comparison of object ranking methods', In Fürnkranz and Hüllermeier [12], 181–201.

[15] R. Kohavi, 'Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid', in *Proceedings of the 2nd International Conference On Knowledge Discovery And Data Mining*, pp. 202–207. AAAI Press, (1996).

[16] P. Paulsen and J. Fürnkranz, 'A moderately successful attempt to train chess evaluation functions of different strengths'. In C. Thurau, K. Driessens, and O. Missura (eds.) *Proceedings of the ICML-10 Workshop on Machine Learning and Games*, Haifa, Israel, (2010).

[17] J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1993.

[18] A. Reinefeld, 'An improvement to the scout tree-search algorithm', *International Computer Chess Association Journal*, **6**(4), 4–14, (December 1983).

[19] A. L. Samuel, 'Some studies in machine learning using the game of checkers', *IBM Journal on Research and Development*, **3**, 210–229, (1959).

[20] Y. Singer and N. Srebro, 'Pegasos: Primal estimated sub-gradient solver for SVM', In Z. Ghahramani (ed.) *Proceedings of the 24th International Conference on Machine Learning (ICML-07)*, pp. 807–814, (2007).

[21] S. S. Skiena, 'An overview of machine learning in computer chess', *International Computer Chess Association Journal*, **9**(1), 20–28, (1986).

[22] R. S. Sutton, 'Learning to predict by the methods of temporal differences', *Machine Learning*, **3**, 9–44, (1988).

[23] G. Tesauro, 'Practical issues in temporal difference learning', *Machine Learning*, **8**, 257–278, (1992).

[24] G. Tesauro, 'Programming backgammon using self-teaching neural nets', *Artificial Intelligence*, **134**(1-2), 181–199, (January 2002). Special Issue on Games, Computers and Artificial Intelligence.

[25] S. Vembu and T. Gärtner, 'Label ranking algorithms: A survey', In Fürnkranz and Hüllermeier [12], 45–64.