Learning Dissimilarities by Ranking: From SDP to QP

Keywords: metric learning, ranking, ordinal regression, kernel, hyperkernel

Abstract

We consider the problem of learning dissimilarities between points via formulations which preserve a specified ordering between points rather than the numerical values of the dissimilarities. *Dissimilarity ranking (d-ranking)* learns from instances like "A is more similar to B than C is to D" or "The distance between E and F is larger than that between G and H". Three formulations of d-ranking problems are presented and new algorithms are presented for two of them, one by semidefinite programming (SDP) and one by quadratic programming (QP). Among the novel capabilities of these approaches are out-of-sample prediction and scalability to large problems.

1. Introduction

Ranking or sometimes referred as *ordinal regression*, is a statistical learning problem which gained much attention recently (Cohen et al., 1998; Herbrich et al., 1999; Joachims, 2002). This problem learns from relative comparisons like "A ranks lower than B" or "C ranks higher than D". The goal is to learn an explicit or implicit function which gives *ranks* over an sampling space $\mathbb{X}$. In most of these tasks, the sampled instances to be ranked are vector-valued data in $\mathbb{R}^D$, while the ranks are real numbers which can be either discrete or continuous. If the problem is to learn a real valued ranking function, it can be stated as: given a set $\mathcal{S}$ of pairs $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S}$ (which indicates that the rank of $\mathbf{x}_i$ is lower than $\mathbf{x}_j$), learn a real valued $f : \mathbb{X} \to \mathbb{R}$ that satisfies $f(\mathbf{x}_m) < f(\mathbf{x}_n)$ if the rank of $\mathbf{x}_m$ is lower than $\mathbf{x}_n$.

In this paper we investigate a special ranking problem: *dissimilarity ranking* (*d-ranking*). Unlike ranking, this problem learns from instances like "A is more similar to B than C is to D" or "The distance between E and F is larger than that between G and H". Note that the dissimilarities here are not necessarily distances. Other than real vectors in conventional ranking problems, the data to be ranked here are dissimilates of pairwised data vectors. This problem can be stated as: learning an explicit or implicit function which gives ranks over a space of dissimilarities $d(\mathbb{X}, \mathbb{X}) \in \mathbb{R}$. Based on different requirements of applications, this learning problem can have various formulations. We will present some of them in Section 2.

D-ranking can be regarded as a special instance of dissimilarity learning (or metric learning). Different dissimilarity learning methods have different goals. We highlight some previous work as below.

- In metric learning methods (Hastie & Tibshirani, 1996; Xing et al., 2002), the purpose of learning a proper Mahalanobis distance is to achieve better class/cluster separations.
- In kernel learning methods (Lanckriet et al., 2004; Micchelli & Pontil, 2005), learning a proper kernel is equivalent to learning a good inner-product function which introduces a dissimilarity in the input space. The purpose is to maximize the performance of a kernel-based learning machine.
- Multidimensional scaling (MDS) (Borg & Groenen, 2005) and Isomap (Tenenbaum et al., 2000) can also be regarded as learning an implicity function $f : \mathbb{R}^D \to \mathbb{R}^L$. The purpose of learning an embedding is to preserve distances in a low-dimensional Euclidean space $\mathbb{R}^L$.

In our d-ranking problems, the purpose of learning a proper dissimilarity is to *preserve the ranks* of dissimilarities, not the absolute values of them (which is the case in MDS and Isomap). For example, if we know that "The distance between A and B is smaller than that between C and D", the problem can be formulated as: find a dissimilarity function d, such that $d(A, B) < d(C, D)$.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Unlike conventional learning and ranking problems, d-ranking hasn't received intensive studies in previous research. One of the most important related work is the nonmetric multidimensional scaling (NMDS) (Borg & Groenen, 2005). Given a symmetric proximity (similarity or dissimilarity) matrix $\Delta = [\delta_{mn}]$, NMDS tries to find a low dimensional embedding space $\mathbb{R}^L$ such that $\forall \mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k, \mathbf{x}_l \in \mathbb{R}^L$, $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 < \|\mathbf{x}_k - \mathbf{x}_l\|_2^2 \Leftrightarrow \delta_{ij} < \delta_{kl}$. NMDS was recently extended to the generalized NMDS (GNMDS) (Agarwal, 2007). GNMDS does not need to know the absolute values of proximities δ_{mn}. Instead it only need a set $\mathcal{S}$ of quadruples $(i, j, k, l) \in \mathcal{S}$, which indicate that $\delta_{ij} < \delta_{kl}$.

Both NMDS and GNMDS learn an embedding space instead of learning an explicit ranking function, thus they are unable to handle out-of-sample problems. Schultz et. al. gave a solution to these problems by proposing to learn a distance metric from relative comparisons (Schultz & Joachims, 2003). They choose to learn a Mahalanobis distance which can preserve ranks of distances. Since the learned distance functions are parameterized, they can be used to handle new samples. The proposed formulation was solved in a similar manner as SVM. Nonetheless, the regularization term was not well justified.

Many applications in biology, computer vision, web search, social science etc. can be put into the framework of d-ranking problems. Take document classification as an instance. Without adequate domain knowledge, it is hard to accurately determine the quantitative dissimilarities between two documents. However, comparing the dissimilarities between every three or four documents can be easily done, either automatically or manually. Generally speaking, d-ranking is especially useful when the quantized dissimilarities are not reliable.

In Section 2, we propose three formulations of d-ranking problems. Section 3 gives the numerical solutions for solving d-ranking by SDP. Section 4 shows how to solve d-ranking by QP. The proposed methods are evaluated in Section 5. Section 6 concludes the paper.

2. Three Formulations of D-Ranking

D-ranking problems can have various formulations depending on specific requirements or settings of applications. Next we will give three formulations.

Formulation 2.1. ***(F1) Inputs****: a set $\mathcal{S}$ of ordered quadruples $(i, j, k, l) \in \mathcal{S}$, indicating that $d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq d(\boldsymbol{x}_k, \boldsymbol{x}_l)$, where $d(\cdot, \cdot)$ is a fixed but unknown dissimilarity function;* ***Outputs****: coefficients of embedded samples $\boldsymbol{x}_i', \boldsymbol{x}_j', \boldsymbol{x}_k', \boldsymbol{x}_l' \in \mathbb{R}^L$;* ***Criteria****: $(i, j, k, l) \in \mathcal{S} \Leftrightarrow \|\boldsymbol{x}_i' - \boldsymbol{x}_j'\|_2^2 \leq \|\boldsymbol{x}_k' - \boldsymbol{x}_l'\|_2^2$.*

As proposed by Agarwal et. al. (Agarwal, 2007), in **F1** we neither assume any geometry of the input space, nor assume any form of dissimilarities in it. We do not need to know the coefficients of input samples. Only ordering information is provided. Nonetheless we assume a Euclidean metric in the embedding space, which is often of low dimensions (e.g. $L = 2$, or 3). As shown in Section 3, **F1** can be formed as a problem of semidefinite programming (SDP).

Formulation 2.2. ***(F2) Inputs****: a set $\mathcal{S}$ of ordered quadruples $(i, j, k, l) \in \mathcal{S}$, indicating that $d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq d(\boldsymbol{x}_k, \boldsymbol{x}_l)$, where $d(\cdot, \cdot)$ is a fixed but unknown dissimilarity function; corresponding coefficients in the input Euclidean space $\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k, \boldsymbol{x}_l \in \mathbb{R}^D$;* ***Outputs****: dissimilarity functions $\hat{d}(\cdot, \cdot) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$;* ***Criteria****: $(i, j, k, l) \in \mathcal{S} \Leftrightarrow \hat{d}(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \hat{d}(\boldsymbol{x}_k, \boldsymbol{x}_l)$.*

Unlike learning an embedding space as in **F1**, **F2** learns an explicit dissimilarity function $\hat{d}(\cdot, \cdot)$ which preserves the ranks of dissimilarities. We will show in Section 4 that **F2** can be handled in a very similar manner as support vector machines, where the quadratic programming (QP) problem can be solved efficiently by specialized sequential optimization methods. If in some cases we need to find a low dimensional Euclidean embedding of the input samples, we can then use the classical multidimensional scaling (MDS) to preserve the learned dissimilarities.

Formulation 2.3. ***(F3) Inputs****: a set $\mathcal{S}$ of ordered quadruples $(i, j, k, l) \in \mathcal{S}$, indicating that $d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq d(\boldsymbol{x}_k, \boldsymbol{x}_l)$, where $d(\cdot, \cdot)$ is a fixed but unknown dissimilarity function; corresponding coefficients in the input Euclidean space $\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k, \boldsymbol{x}_l \in \mathbb{R}^D$;* ***Outputs****: projection function $f : \mathbb{R}^D \to \mathbb{R}^L$, $\boldsymbol{x}_i', \boldsymbol{x}_j', \boldsymbol{x}_k', \boldsymbol{x}_l' \in \mathbb{R}^L$;* ***Criteria****: $(i, j, k, l) \in \mathcal{S} \Leftrightarrow \|\boldsymbol{x}_i' - \boldsymbol{x}_j'\|_2^2 \leq \|\boldsymbol{x}_k' - \boldsymbol{x}_l'\|_2^2$.*

Although we formulate **F3** as a function learning problem, currently we have not found any efficient method to solve it. This formulation will remains as our future work.

3. Solving F1 by SDP

F1 was studied by Agarwal et. al. (Agarwal, 2007). The authors proposed GNMDS which can be solved as

a SDP, as shown in Eq.(1).

$$\begin{aligned} &\text{GNMDS:} \\ \min\ & \sum_{ijkl\in\mathcal{S}} \xi_{ijkl} + \lambda \mathrm{tr(K)}, \\ \text{s.t. } & (K_{kk} - 2K_{kl} + K_{ll}) - (K_{ii} - 2K_{ij} + K_{jj}) \\ & + \xi_{ijkl} \geq 1, \\ & \sum_{ab} K_{ab} = 0,\ \xi_{ijkl} \geq 0,\ K \succeq 0, \\ & \text{for all } (i,j,k,l) \in \mathcal{S}. \end{aligned} \tag{1}$$

The main idea of GNMDS is to learn a positive semidefinite Gram matrix $K = X^T X$ which can be eigen-decomposed to recover the embedded samples. The relation between Euclidian distances and the Gram matrix is used:

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = K_{ii} - 2K_{ij} + K_{jj}. \tag{2}$$

Nonetheless, the constraints that contain order information of dissimilarities are not sufficient to determine a unique K, since any rotation, translation or scaling can also satisfies these constraints. To reduce these ambiguities, they use $\sum_{ab} K_{ab} = 0$ to center all the embedded samples at the origin.

It is preferable in many applications to find low dimensional embedding spaces, e.g. 2D or 3D Euclidean space. Thus a low-rank K is desired. Unfortunately, minimizing rank(K) subject to linear inequality constraints is NP-Hard (Vandenberghe & Boyd, 1996). Thus the objective function is relaxed heuristically as minimizing trace(K), which is a convex envelope of the rank.

Figure 1 shows the result of GNMDS on a toy problem. The inputs are 990 pairwise ranks of distances between 10 European cities. The outputs are the recovered 2D coefficients. It can be observed that the recovered locations of the cities do not correspond to the true locations. Actually only 789 out of 990 pairs of ranks are preserved by the learned 2D embedding, i.e. 20.3% error rate. Figure 2 shows the 10 sorted eigenvalues of the Gram matrix K. Although the original space is a 2D Euclidean space, the first 2 eigenvalues only account for 49.4% of the total variation.

There are at least two reasons that account for the poor performance of GNMDS. Firstly, there is no guarantee on the quality of the solution of the relaxed problem compared with the original problem. There may exist some higher dimensional spaces which satisfy all the constants while have smaller traces than lower dimensional spaces. Secondly, due to the introduction of

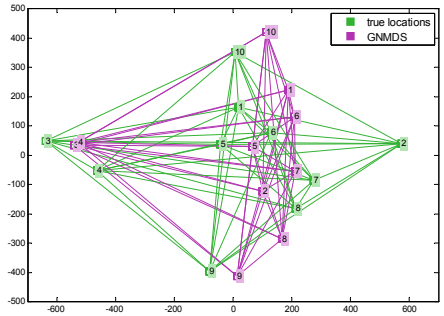

Figure 1. d-ranking toy problem: locations of ten European cities. Purple: true locations. Green: locations recovered by GNMDS. All the coefficients have been scaled before plotting.

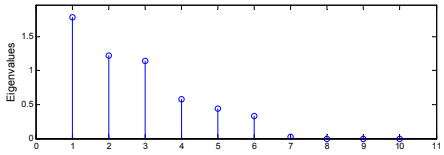

Figure 2. 10 sorted eigenvalues of the Gram matrix K learned by GNMDS.

slack variables ξ_{ijkl} in the inequality constraints, the learned embedding tends to push all the samples to the same point. The first problem can be solved by introducing better heuristics of the convex envelope of the rank. The second problem can be solved by the following slight modification of GNMDS:

$$\begin{aligned} &\text{Modified GNMDS:} \\ \min\ & \sum_{ijkl\in\mathcal{S}} \xi_{ijkl} + \lambda \mathrm{tr(K)}, \\ \text{s.t. } & (K_{kk} - 2K_{kl} + K_{ll}) - (K_{ii} - 2K_{ij} + K_{jj}) \\ & - \xi_{ijkl} \geq 1, \\ & \sum_{ab} K_{ab} = 0,\ \xi_{ijkl} \geq 0,\ K \succeq 0, \\ & \text{for all } (i,j,k,l) \in \mathcal{S}. \end{aligned} \tag{3}$$

The modified GNMDS just changes the slack variables from $+\xi_{ijkl}$ to $-\xi_{ijkl}$. This simple trick can ensure that all the differences between distances k,l and i,j are larger than 1, thus pulls the embedding samples apart. Figure 3 shows toy problem solved by the modified GNMDS. The recovered samples are closer to the true locations than those in Figure 1. There are 850 out of 990 pairs of ranks correctly preserved, i.e. 14.14% error rate, which is 6% lower than GNMDS. Figure

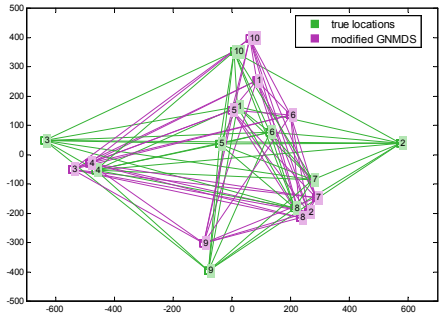

Figure 3. d-ranking toy problem: locations of ten European cities. Purple: true locations. Green: locations recovered by modified GNMDS. All the coefficients have been scaled before plotting.

4 shows the 10 eigenvalues of K learned by modified GNMDS. The first 2 eigenvalues account to 69.8% of the total variant, which is 20% higher than GNMDS.

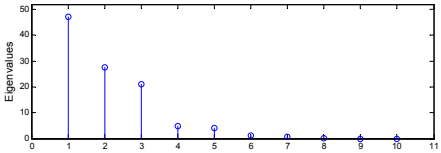

Figure 4. 10 sorted eigenvalues of the Gram matrix K learned by modified GNMDS.

4. Solving F2 by QP

As introduced in Section 2, instead of learning an embedding as in **F1**, a dissimilarity function $d(\mathbf{x}_i, \mathbf{x}_j) : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is learned in **F2**, such that all the training ranks between $d(\cdot, \cdot)$ are preserved, and can generalize to new samples. This is indeed a dissimilarity learning problem.

Many previous metric learning methods (Hastie & Tibshirani, 1996; Goldberger et al., 2004; Kwok & Tsang, 2003) try to learn an alternative dissimilarity function by replacing the Euclidean metric with an properly learnt Mahalanobis metric, either globally or locally.

In this section we propose the *d-ranking Vector Machine* (*d-ranking-VM*) method. Unlike metric learning methods, d-ranking-VM is explicitly regularized. Thus we can have a full control over the complexity of $d(\mathbf{x}_i, \mathbf{x}_j)$. D-ranking-VM utilizes the technique of hyperkernel learning (Ong et al., 2005) which was originally proposed for learning a proper kernel.

D-ranking-VM is formulated as the following optimization problem:

$$\begin{aligned} &\text{d-ranking-VM (primal):} \\ \min\ & \frac{1}{N}\sum_{ijkl\in\mathcal{S}} \xi_{ijkl} + \lambda\|d\|^2_{\underline{\mathbb{H}}}, \\ \text{s.t. }\ & d(\mathbf{x}_k, \mathbf{x}_l) - d(\mathbf{x}_i, \mathbf{x}_j) - \xi_{ijkl} \geq 1, \\ & \xi_{ijkl} \geq 0, \\ & \text{for all } (i,j,k,l) \in \mathcal{S}. \end{aligned} \tag{4}$$

where $N = |\mathcal{S}|$. $\underline{\mathbb{H}}$ is a hyper-reproducing kernel Hilbert space (hyper-RKHS) from which the function $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is drawn.

Like the representer theorem in RKHS (Kimeldorf & Wahba, 1971), there is also a representer theorem in hyper-RHKS (see (Ong et al., 2005) or (Kondor & Jebara, 2006) for the theorem and proofs):

$$d(\underline{\mathbf{x}}) = \sum_{p=1}^{M} c_p \underline{K}(\underline{\mathbf{x}}_p, \underline{\mathbf{x}}), \tag{5}$$

where $\underline{K}$ is a semidefinite hyperkernel, $\underline{\mathbf{x}}$ denotes a pair of samples $(\mathbf{x}_i, \mathbf{x}_j)$, and M is the number of distinct dissimilarity pairs provided by the training rank data $\mathcal{S}$. We denote the set of dissimilarity pairs as $\mathcal{D}$, and $M = |\mathcal{D}|$. Normally we have $M > N$ (the discussion of M and N is given in Section 5.1).

Substitute Eq.(5) into (4), we can change the primal problem to the following form:

$$\begin{aligned} \min\ & \frac{1}{N}\sum_{p\in\mathcal{S}} \xi_p + \lambda C^T \underline{K} C, \\ \text{s.t. }\ & \sum_{p=1}^{M} c_p \underline{K}(\underline{\mathbf{x}}_p; \mathbf{x}_k, \mathbf{x}_l) - \sum_{p=1}^{M} c_p \underline{K}(\underline{\mathbf{x}}_p; \mathbf{x}_i, \mathbf{x}_j) \\ & - \xi_p \geq 1, \\ & \xi_p \geq 0, \text{for all } p \in \mathcal{S}, \end{aligned} \tag{6}$$

where $C \in \mathbb{R}^M$ is a vector with the ith element being c_p, and $\underline{K} \in \mathbb{R}^{M\times M}$ is the hyper-kernel matrix.

The dual problem of (6) can be derived by using the Lagrangian technique. The solution to this optimization problem is given by the saddle point of the Lagrangian function:

$$\begin{aligned} L(C, \xi_p, \alpha_p, \zeta_p) = & \frac{1}{N}\sum_{p\in\mathcal{S}} \xi_p + \lambda C^T \underline{K} C - \sum_{p=1}^{N} \zeta_p \xi_p \\ & + \sum_{p=1}^{N} \alpha_p \left\{ \sum_{s=1}^{M} c_s \left[\underline{K}(\underline{\mathbf{x}}_s; \mathbf{x}_i, \mathbf{x}_j) - \underline{K}(\underline{\mathbf{x}}_s; \mathbf{x}_k, \mathbf{x}_l) \right] + 1 + \xi_p \right\}, \end{aligned} \tag{7}$$

where ζ_p and α_p are non-negative Lagrange multipliers. The primal problem is convex, thus there exist a strong duality between the primal and the dual. Utilizing the KKT optimality condition, we have:

$$\frac{\partial L}{\partial C} = 2\lambda \underline{K} C + \underline{K}(P-Q)A = 0, \tag{8}$$

and

$$\frac{\partial L}{\partial \xi_p} = \frac{1}{N} + \alpha_p - \zeta_p = 0, \tag{9}$$

where $A \in \mathbb{R}^N$ is a vector with the pth element being α_p. $P, Q \in \mathbb{R}^{M\times N}$ are two matrices with contain the rank information. Each column $p_{x\cdot}(x = 1 \dots N)$ of P and each column $q_{x\cdot}(x = 1 \dots N)$ of Q only contain one 1 and $M-1$ 0s. For example, if the rth training quadruples in $\mathcal{S}$ is (i, j, k, l), which means that $d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_k, \mathbf{x}_l)$, and if the pair (i, j) is the mth element in $\mathcal{D}$, while (k, l) is the $n$the element in $\mathcal{D}$, then $p_{rm} = 1$ and $q_{rn} = 1$.

From Eq.(8) and (9) we have:

$$C = \frac{(Q-P)A}{2\lambda}, \tag{10}$$

and

$$\alpha_p = \zeta_p - \frac{1}{N} \tag{11}$$

Substitute Eq.(10) and (11) into (6), we arrive at the following *dual* problem for d-ranking-VM:

$$\begin{aligned} &\text{d-ranking-VM (dual):} \\ &\max \sum_{p=1}^{N} \alpha_p - \frac{A^T(Q-P)^T \underline{K}(Q-P)A}{4\lambda}, \\ &\text{s.t. } \alpha_p \geq 0, \\ &\text{for all } (i,j,k,l) \in \mathcal{S}. \end{aligned} \tag{12}$$

This problem is a quadratic programming (QP) problem which shares a similar form as SVM. Thus the sequential optimization techniques of SVM can be readily employed for d-ranking-VM. To perform testing, we can use the learnt dissimilarity function in Eq.(5) and make pairwise comparisons.

An important problem for kernel learning methods is the selection of proper kernels. This problem also exists in hyperkernel learning methods. Here we propose some examples of hyperkernels, which are hyperextensions of Gaussian RBF kernels and polynomial kernels. The construction of these hyperkernels are based on the following proposition.

Proposition 4.1. *Let $k_a(\cdot,\cdot)$ and $k_b(\cdot,\cdot)$ be positive definite kernels, then $\forall \boldsymbol{x}_1, \boldsymbol{x}_1^{'}, \boldsymbol{x}_2, \boldsymbol{x}_2^{'} \in \mathbb{X}$, and $\forall \alpha, \beta > 0$, $\big(k_a(\boldsymbol{x}_1, \boldsymbol{x}_2)\big)^{\alpha}\big(k_b(\boldsymbol{x}_1^{'}, \boldsymbol{x}_2^{'})\big)^{\beta}$ or $\alpha k_a(\boldsymbol{x}_1, \boldsymbol{x}_2) + \beta k_b(\boldsymbol{x}_1^{'}, \boldsymbol{x}_2^{'})$ can give a hyperkernel $\underline{k}$.*

Proof. See appendix. □

Example 4.2. ***(Gaussian symmetric product hyperkernel)*** *Let k_a and k_b be the same Gaussian RBF kernel $k(\boldsymbol{x}, \boldsymbol{x}^{'}) = \exp\big(-\frac{\|\boldsymbol{x}-\boldsymbol{x}^{'}\|^2}{2\sigma^2}\big)$, and let $\alpha = \beta = 1$, the Gaussian symmetric product hyperkernel is given by:*

$$\begin{aligned} &\underline{k}\big((\boldsymbol{x}_1, \boldsymbol{x}_1^{'}), (\boldsymbol{x}_2, \boldsymbol{x}_2^{'})\big) = k(\boldsymbol{x}_1, \boldsymbol{x}_1^{'})k(\boldsymbol{x}_2, \boldsymbol{x}_2^{'}) \\ &= \exp\bigg(-\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_1^{'}\|^2 + \|\boldsymbol{x}_2 - \boldsymbol{x}_2^{'}\|^2}{2\sigma^2}\bigg) \end{aligned} \tag{13}$$

Example 4.3. ***(Gaussian symmetric sum hyperkernel)*** *Under the same conditions as Example 4.2, we can construct the Gaussian symmetric sum hyperkernel as:*

$$\begin{aligned} &\underline{k}\big((\boldsymbol{x}_1, \boldsymbol{x}_1^{'}), (\boldsymbol{x}_2, \boldsymbol{x}_2^{'})\big) = k(\boldsymbol{x}_1, \boldsymbol{x}_1^{'}) + k(\boldsymbol{x}_2, \boldsymbol{x}_2^{'}) \\ &= \exp\bigg(-\frac{\|\boldsymbol{x}_1 - \boldsymbol{x}_1^{'}\|^2}{2\sigma^2}\bigg) + \exp\bigg(-\frac{\|\boldsymbol{x}_2 - \boldsymbol{x}_2^{'}\|^2}{2\sigma^2}\bigg) \end{aligned} \tag{14}$$

Example 4.4. ***(polynomial symmetric product hyperkernel)*** *Let k_a and k_b be the same polynomial kernel $k(\boldsymbol{x}, \boldsymbol{x}^{'}) = \big(\langle \boldsymbol{x}, \boldsymbol{x}^{'}\rangle + q\big)^p$, and let $\alpha = \beta = 1$, we can construct the polynomial symmetric product hyperkernel as:*

$$\underline{k}\big((\boldsymbol{x}_1, \boldsymbol{x}_1^{'}), (\boldsymbol{x}_2, \boldsymbol{x}_2^{'})\big) = \big(\langle \boldsymbol{x}_1, \boldsymbol{x}_1^{'}\rangle + q\big)^p \big(\langle \boldsymbol{x}_2, \boldsymbol{x}_2^{'}\rangle + q\big)^p \tag{15}$$

Example 4.5. ***(polynomial symmetric sum hyperkernel)*** *Under the same conditions as Example 4.4, we can construct the polynomial symmetric sum hyperkernel as:*

$$\underline{k}\big((\boldsymbol{x}_1, \boldsymbol{x}_1^{'}), (\boldsymbol{x}_2, \boldsymbol{x}_2^{'})\big) = \big(\langle \boldsymbol{x}_1, \boldsymbol{x}_1^{'}\rangle + q\big)^p + \big(\langle \boldsymbol{x}_2, \boldsymbol{x}_2^{'}\rangle + q\big)^p \tag{16}$$

5. Experiments

The proposed d-ranking methods in Section 3 and Section 4 are evaluated by several experiments.

5.1. Obtaining Ranks of Pairs from Data

To test our methods, we need to obtain pairwise distance ranks. This can be done in many ways. Generally speaking, for a problem of n data samples, the total number of available distance pairs are $M = C_n^2 = \frac{n(n-1)}{2}$ (if we take $d(\mathbf{x}_i, \mathbf{x}_j)$ and $d(\mathbf{x}_j, \mathbf{x}_i)$ as the same

distance). The total number of pairwise distance ranks are $N = C_M^2 = \frac{M(M-1)}{2} = \frac{n^4}{8} - \frac{n^3}{4} - \frac{n^2}{8} + \frac{n}{4}$ (if we take $d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_k, \mathbf{x}_l)$ and $d(\mathbf{x}_k, \mathbf{x}_l) > d(\mathbf{x}_i, \mathbf{x}_j)$ as the same rank pair). Table 1 gives some examples of the relation between n and N. When n grows, the number of rank constraints will increase dramatically. Even solving a problem of $n > 100$ will be impossible for some optimization solvers.

Table 1. Some examples of N v.s. n.

n	2	3	4	10	20	50	100	1000
N	0	3	15	990	17955	749700	12248775	1.2475e+11

Here we reduce N by considering order transitivities, i.e. if $A > B$ and $B > C$, then the rank pair $A > C$ can be ignored (automatically satisfied) in the optimization constraints. The method is very simple. Firstly we sort the M distances decreasingly. Then we take the adjacent two distances to form one distance rank pair. By doing this, N can be reduced to $\frac{n^2}{2} - \frac{n}{2} - 1$. This is the maximum number of N which carries full rank information of all the distances. Of course in some applications, the rank information is not be fully given, and $N < \frac{n^2}{2} - \frac{n}{2} - 1$.

We test our method on three data sets: 1) 2D locations of 109 largest cities in the continental US; 2) 100 images of handwritten digits "3" and "5" from the USPS database, each of size 16×16; 3) 126 face images of 4 people from the UMist database, each of size 112×92.

For GNMDS and modified GNMDS methods, all ranks of distance pairs are fed to a SDP solver, and the recovered 2D embeddings are plotted. We used SeDuMi (Strum, 1999) to get the results given in the following subsection. For d-ranking-VM, LibSVM (Chang & Lin, 2001) is employed as our QP solver. It is implemented by employing the sequential minimal optimization (SMO) technique. The learned dissimilarities are used as a "pseudo-distances", and are fed to the classical MDS. The recovered 2D embeddings are then plotted.

5.2. Results of US Cities

In this data set, $n = 109$ and $N = 5885$. Every location of the cities is given by a 2D coefficient. Figure 5 shows the true locations. Figure 6 shows the results given by GNMDS.

It can be observed that GNMDS cannot correctly recover the embedding based on distance ranks. Most of the embedded samples are pushed to a line. 50.3% of the distances ranks are preserved, which are the results of randomness. This gives an evidence to the analysis in Section 3. Figure 7 shows the result given by the modified GNMDS. The recovered embedding roughly reflects the geometry of the cities. 74.5% of the distances ranks have been preserved. Since the distance information is not provided, there is no hope to match the true locations exactly.

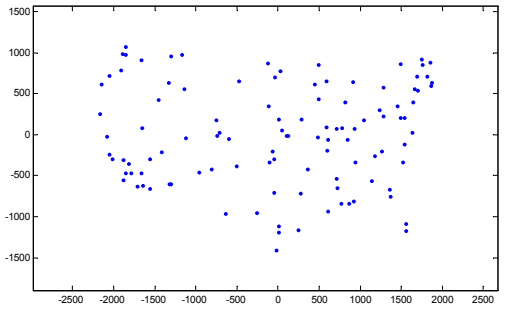

Figure 5. Locations of 109 largest cities in the continental United States.

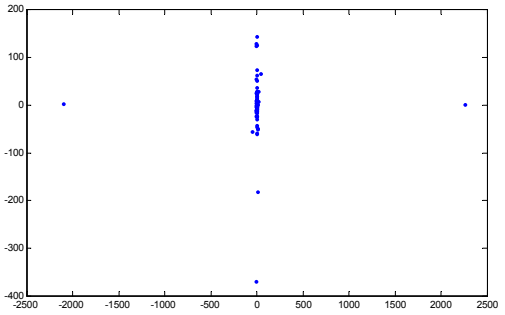

Figure 6. Recovered locations of 109 US cities given by GNMDS.

Figure 8 shows the results given by d-ranking-VM, where $\lambda = 10$, and the Gaussian symmetric product hyperkernel is used, with $\sigma = 15$. 97.9% of the distances ranks are preserved.

Table 2 shows the runtime of the above experiments.

Table 2. Runtime comparison for the three d-ranking methods.

Method	Runtime(minutes)
GNMDS	124
modified GNMDS	71
d-ranking-VM	4.5

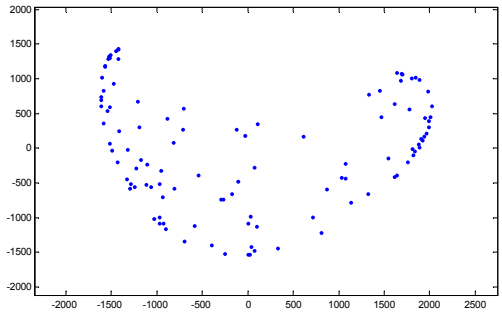

Figure 7. Recovered locations of 109 US cities given by the modified GNMDS.

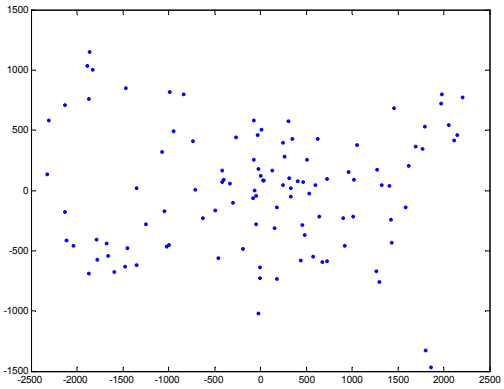

Figure 8. Recovered locations of 109 US cities given by the d-ranking-VM, using Gaussian symmetric product hyperkernel.

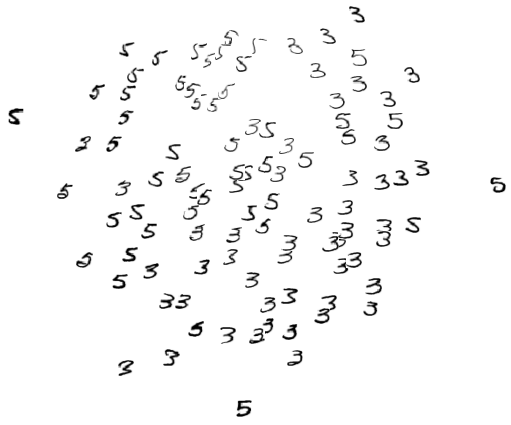

Figure 9. Recovered locations of USPS handwritten digits "3" and "5" given by d-ranking-VM.

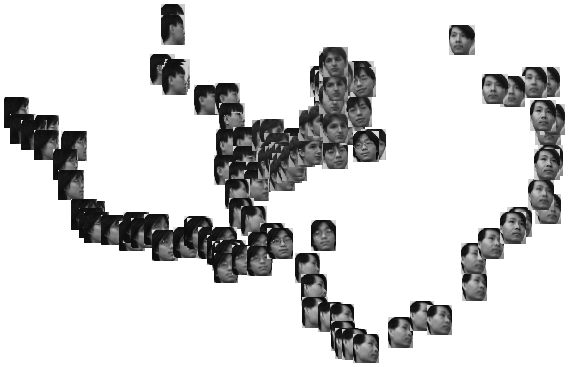

Figure 10. Recovered locations of UMist human faces given by d-ranking-VM.

5.3. Results of USPS Handwritten Digits

In this data set, $n = 100$ and $N = 4949$. The dimension every data sample is $16 \times 16 = 256$. Figure 9 shows the recovered 2D results given by RnakD-VM.

5.4. Results of UMist Human Faces

In this data set, $n = 126$ and $N = 7874$. The dimension every data sample is $112 \times 92 = 10304$. Figure 10 shows the recovered 2D results given by RnakD-VM.

6. Discussions and Conclusions

We have presented three d-ranking formulations, and give numerical solutions for two of them, namely solving d-ranking by SDP and solving d-ranking by QP. Each of them has its advantages and shortcomings. We list some pros and cons from different perspectives:

- Pros for d-ranking by SDP (GNMDS and the modified version): It can recover low dimensional embedding directly. Only ordering information is needed. There is no need to know the values of sample coefficients.
- Cons for d-ranking by SDP (GNMDS and the modified version): Solving SDP is hard, especially for large scale problems. Even sophistries SDP solver can only solve $N < 10^3$ problems with the number of constraints less than 10^5. It cannot be used to predict unseen samples.
- Pros for d-ranking by QP (d-ranking-VM): Solving QP in our case is much easier than SDP, since it can be converted to a similar form as SVM. Using sequential methods (SMO), can solve $N > 10^5$ problems. The learn dissimilarity function can be used to predict unseen samples.
- Cons for d-ranking by QP (d-ranking-VM): It can-

not recover low-dimensional embedding explicitly. One needs to use MDS or other embedding methods after learning the dissimilarities. Learning dissimilarity measure needs to know the coefficients of original samples. Like kernel methods, how to choose a good hyperkernel is crucial in solving a specific problem.

To our knowledge, this is the first work which brings out-of-sample prediction capability and large-scale scalability to d-ranking problems. Note that the technique of d-ranking-VM can also be employed in solving distances preserving problems. We will investigate the regularization properties and evaluate the performances of different hyperkernels in the following research. Finding a numerical solution for formulation **F3** will also be our future work.

7. Appendix: Proof of Proposition 4.1

We need to prove that $\underline{k}$ is a kernel on $\mathbb{X}^2$.

Denote $\bigotimes$ as the Kronecker product of two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$:

$$A \bigotimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}, \qquad (17)$$

and define $\bigoplus$ as:

$$A \bigoplus B = \begin{bmatrix} a_{11}\mathbb{1} + B & \cdots & a_{1n}\mathbb{1} + B \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbb{1} + B & \cdots & a_{mn}\mathbb{1} + B \end{bmatrix}, \qquad (18)$$

where $\mathbb{1}$ is a matrix of all ones.

Denote the kernel matrix of $k_a(\mathbf{x}_1, \mathbf{x}_2)$ as K_a, and that of $k_b(\mathbf{x}_1^{'}, \mathbf{x}_2^{'})$ as K_b. Denote the hyperkernel matrix of $\underline{k}$ as $\underline{K}$.

It is easy to verify that we can construct $\underline{K} = K_a \bigotimes K_b$. Since K_a and K_b are positive definite, their eigenvalues μ_a and μ_b are positive. Thus the eigenvalues of $\underline{K}$: $v_{ij} = \alpha\beta\mu_{ai}\mu_{bj}$ are also positive. A symmetric matrix $\underline{K}$ with positive eigenvalues is positive definite. Thus $\underline{k} = \big(k_a(\mathbf{x}_1, \mathbf{x}_2)\big)^{\alpha}\big(k_b(\mathbf{x}_1^{'}, \mathbf{x}_2^{'})\big)^{\beta}$ is a valid hyperkernel.

We can also verify that $\alpha K_a \bigoplus \beta K_b = \alpha K_a \bigotimes \mathbb{1} + \beta\mathbb{1} \bigotimes K_b$. Since K_a, K_b and $\mathbb{1}$ are all positive semidefinite and $\alpha, \beta > 0$, $\underline{K} = \alpha K_a \bigoplus \beta K_b$ is positive semidefinite. Thus $\underline{k} = \alpha k_a(\mathbf{x}_1, \mathbf{x}_2) + \beta k_b(\mathbf{x}_1^{'}, \mathbf{x}_2^{'})$ is a valid hyperkernel.

References

Agarwal, S. (2007). Generalized non-metric multidimensional scaling. *AISTATS 07*.

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

Chang, C.-C., & Lin, C.-J. (2001). LIBSVM: a library for support vector machines. Software available at *http://www.csie.ntu.edu.tw/ ∼ cjlin/libsvm*.

Cohen, W. W., Schapire, R. E., & Singer, Y. (1998). Learning to order things. *NIPS*.

Goldberger, J., Roweis, S., Hinton, G., & Salakhutdinov, R. (2004). Neighbourhood Components Analysis. *NIPS*.

Hastie, T., & Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification. *IEEE Trans. PAMI*, *18*, 607–616.

Herbrich, R., Graepel, T., & Obermayer, K. (1999). Large margin rank boundaries for ordinal regression. *Advances in Large Margin Classifiers*, 115–132.

Joachims, T. (2002). Optimizing search engines using clickthrough data. *Proc. of SIGKDD '02*.

Kimeldorf, G., & Wahba, G. (1971). Some Results on Tchebycheffian Spline Functions. *Journal of Mathematical Analysis and Applications*, *33*, 82–75.

Kondor, R., & Jebara, T. (2006). Gaussian and Wishart Hyperkernels. *NIPS*.

Kwok, J. T., & Tsang, I. W. (2003). Learning with Idealized Kernels. *ICML*.

Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L., & Jordan, M. I. (2004). Learning the Kernel Matrix with Semidenite Programming. *Journal of Machine Learning Research*, *5*, 27–72.

Micchelli, C. A., & Pontil, M. (2005). Learning the Kernel Function via Regularization. *Journal of Machine Learning Research*, *6*, 1099–1125.

Ong, C. S., Smola, A. J., & Williamson, R. C. (2005). Learning the Kernel with Hyperkernels. *Journal of Machine Learning Research*, *6*, 1043–1071.

Schultz, M., & Joachims, T. (2003). Learning a Distance Metric from Relative Comparisons. *NIPS*.

Strum, J. F. (1999). Using SeDuMi 1.02, A Matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, *11*.

Tenenbaum, J. B., de Silva, V., & Langford, J. C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, *290*, 2319–2323.

Vandenberghe, L., & Boyd, S. (1996). Semidefinite Programming. *SIAM Review*, *38*, 49–95.

Xing, E. P., Ng, A. Y., Jordan, M. I., & Russell, S. (2002). Distance Metric Learning with Application to Clustering with Side-Information. *NIPS*.