Metric Learning to Rank

Brian McFee BMCFEE@CS.UCSD.EDU
Department of Computer Science and Engineering, University of California, San Diego, CA 92093 USA

Gert Lanckriet GERT@ECE.UCSD.EDU
Department of Electrical and Computer Engineering, University of California, San Diego, CA 92093 USA

Abstract

We study metric learning as a problem of information retrieval. We present a general metric learning algorithm, based on the structural SVM framework, to learn a metric such that rankings of data induced by distance from a query can be optimized against various ranking measures, such as AUC, Precision-at-k, MRR, MAP or NDCG. We demonstrate experimental results on standard classification data sets, and a large-scale online dating recommendation problem.

1. Introduction

In many machine learning tasks, good performance hinges upon the definition of similarity between objects. Although Euclidean distance on raw features provides a simple and mathematically convenient metric, there is often no reason to assume that it is optimal for the task at hand. Consequently, many researchers have developed algorithms to automatically learn distance metrics in supervised settings.

With few exceptions, these *metric learning* algorithms all follow the same guiding principle: a point's *good* neighbors should lie closer than its *bad* neighbors. The exact definitions of *good* and *bad* vary across problem settings, but typically they derive from some combination of proximity and label agreement. In keeping with this principle, metric learning algorithms are often evaluated by testing the accuracy of labels predicted by k-nearest neighbors on held out data.

At a high level, we consider a metric good if, when given a test point q, sorting the training set by increasing distance from q results in good neighbors at the front of the list, and bad neighbors at the end. Viewed in this light, we can cast nearest neighbor prediction as a ranking problem, and the predicted label error rate as a loss function over rankings. Thus, at its core, the metric learning problem is a special case of information retrieval in the *query-by-example* paradigm.

In recent years, many advances have been made in the development of learning algorithms for ranking (Joachims, 2005; Burges et al., 2005; Xu & Li, 2007; Volkovs & Zemel, 2009). Unlike the classification problems typically addressed by metric learning, ranking problems generally lack a single evaluation criterion. Rather, several evaluation measures have been proposed, each capturing a different notion of "correctness." Because rankings are inherently combinatorial objects, these evaluation measures are often non-differentiable with respect to model parameters, and therefore difficult to optimize by learning algorithms. Despite the combinatorial difficulties of ranking problems, there are now several algorithmic techniques for optimizing various ranking evaluation measures (Joachims, 2005; Chakrabarti et al., 2008; Volkovs & Zemel, 2009).

In the present work, we seek to bridge the gap between metric learning and ranking. By adapting techniques from information retrieval, we derive a general metric learning algorithm which optimizes for the true quantity of interest: the permutation of data induced by distances in the learned metric.

Conversely, our parameterization of the ranking function by a distance metric is quite natural for many information retrieval applications, including multimedia recommendation.

The present approach, based on structural SVM (Tsochantaridis et al., 2005), readily supports various ranking evaluation measures under a unified algorithmic framework. The interpretation of metric learning as an information retrieval problem

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

allows us to apply loss at the level of rankings, rather than pairwise distances, and enables the use of more general notions of similarity than those used in previous metric learning algorithms, including asymmetric and non-transitive definitions of relevance.

1.1. Related work

There has been a great deal of research devoted to the design of algorithms for learning an optimal metric in supervised settings. Typically, these algorithms follow the general scheme of learning a (preferably low-rank) linear projection of the data such that distances to a pre-determined set of "good neighbors" is minimized, while "bad neighbor" distances are maximized.

Xing et al. (2003) define the good neighbors as all similarly-labeled points, and solve for the metric by semidefinite programming. Distances for similar pairs of points are upper-bounded by a constant, and dissimilar-pair distances are maximized. This algorithm attempts to map each class into a ball of fixed radius, but does not enforce separation between classes.

Weinberger et al. (2006) define target neighbors as the k closest similarly-labeled points in the original feature space, and forces positive margins between distances to target neighbors and all other (dissimilar) points. This relaxes the constraint of Xing et al. (2003) that all points of a given class must lie close to each-other, and the algorithm performs well in many real-world scenarios. However, as illustrated in Figure 1, the dependence on the original feature space for determining target neighbors can make the algorithm unsuitable for problem domains involving noisy or heterogeneous features: a single corrupted feature can dominate the initial distance calculations, and prevent the algorithm from finding an optimal projection.

Neighborhood components analysis (NCA) (Goldberger et al., 2005) relaxes the problem by maximizing the expected number of correctly retrieved points under a stochastic neighbor selection rule. While this relaxation makes intuitive sense, the resulting optimization is non-convex, and it cannot identify and optimize the top-k nearest neighbors in the learned space. Globerson & Roweis (2006) optimize a similar stochastic neighbor selection rule while attempting to collapse each class to a single point. This idea enforces more regularity on the output space than NCA and leads to a convex optimization problem, but the assumption that entire classes can be collapsed to distinct points rarely holds in practice.

The core of our method is based on the structural SVM framework (Tsochantaridis et al., 2005). We provide a brief overview in Section 2, and discuss ranking-specific extensions in Section 4.

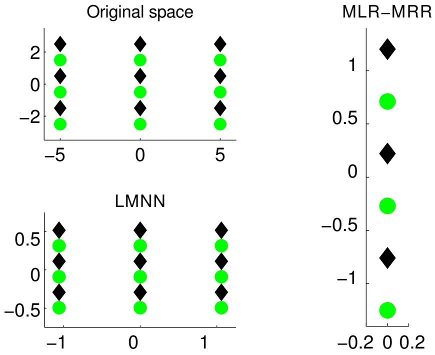

Figure 1. A toy example illustrating the dangers of relying on input features for determining good neighbors. Top-left: A binary-labeled (●, ◆) data set in its native feature space. All discriminative information is contained in the vertical axis, but the scale of the horizontal axis corrupts the selection of *good* neighbors. Bottom-left: LMNN($k = 3$) selects the vertical neighbors, and does not find the optimal projection. Right: Metric learning to rank (MLR) correctly projects onto the vertical axis.

1.2. Preliminaries

Let $\mathcal{X} \subset \mathbb{R}^d$ denote the training set (corpus), with $|\mathcal{X}| = n$. $\mathcal{Y}$ will denote the set of permutations (rankings) of $\mathcal{X}$. For a query q, let $\mathcal{X}_q^+$ and $\mathcal{X}_q^-$ denote the subsets of relevant and irrelevant points in the training set. For a ranking $y \in \mathcal{Y}$ and two points $i, j \in \mathcal{X}$, we will use $i \prec_y j$ ($i \succ_y j$) to indicate that i is placed before (after) j in y.

$W \succeq 0$ will denote a symmetric, positive semi-definite matrix in $\mathbb{R}^{d\times d}$. For $i, j \in \mathbb{R}^d$, we will denote distance under the metric defined by W as $\|i - j\|_W = \sqrt{(i-j)^\mathsf{T} W (i-j)}$. For matrices $A, B \in \mathbb{R}^{d\times d}$, we will denote their Frobenius inner product as $\langle A, B\rangle_F = \mathrm{tr}(A^\mathsf{T} B)$. Finally, $\mathbb{1}[X]$ will denote the 0-1 indicator function on the event X.

2. Structural SVM review

Structural SVM can be viewed as a generalization of multi-class SVM (Crammer & Singer, 2002), where the set of possible prediction outcomes is generalized from labels to structures, *e.g.*, a parse tree, permutation, sequence alignment, etc. (Tsochantaridis et al., 2005). The multi-class SVM formulation of Crammer & Singer (2002) forces margins for each training point $q \in \mathcal{X}$ between the true label y^* and all other labels y:

$$\forall y \neq y^* : \ w_{y^*}^\mathsf{T} q \geq w_y^\mathsf{T} q + 1 - \xi,$$

where $\xi \geq 0$ is a slack variable to allow margin violations on the training set. Similarly, structural SVM applies margins between the true *structure* y^* and all other possible structures y:

$$\forall y \in \mathcal{Y}: \ w^\mathsf{T}\psi(q, y^*) \geq w^\mathsf{T}\psi(q, y) + \Delta(y^*, y) - \xi. \quad (1)$$

Here, $\psi(q, y)$ is a vector-valued joint feature map which characterizes the relationship between an input q and an output structure y. (This notation subsumes the class-specific discriminant vectors of multi-class SVM.) Unlike class labels, two distinct structures (y^*, y) may exhibit similar accuracy, and the margin constraint should reflect this. To support more flexible notions of structural correctness, the margin is set to $\Delta(y^*, y)$: a non-negative loss function defined between structures, which is typically bounded in $[0, 1]$.

For a test query $\hat{q}$ in multi-class SVM, the predicted label y is that which maximizes $w_y^\mathsf{T}\hat{q}$, *i.e.*, the label with the largest margin over other labels. Analogously, structural predictions are made by finding the structure y which maximizes $w^\mathsf{T}\psi(\hat{q}, y)$. The prediction algorithm must be able to efficiently use the learned vector w when computing the output structure y. As we will see in Sections 2.2 and 3, this is easily accomplished in general ranking, and specifically in metric learning.

2.1. Optimization

Note that the set $\mathcal{Y}$ of possible output structures is generally quite large (*e.g.*, all possible permutations of the training set), so enforcing all margin constraints in (1) may not be feasible in practice. However, cutting planes can be applied to efficiently find a small working set of active constraints which are sufficient to optimize w within some prescribed tolerance (Tsochantaridis et al., 2005).

The core component of the cutting plane approach is the *separation oracle*, which given a fixed w and input point q, outputs the structure y corresponding to the margin constraint for q which is most violated by w:

$$y \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} w^\mathsf{T}\psi(q, y) + \Delta(y^*, y). \quad (2)$$

Intuitively, this computes the structure y with simultaneously large loss $\Delta(y^*, y)$ and margin score $w^\mathsf{T}\psi(q, y)$: in short, the weak points of the current model w. Adding margin constraints for these structures y efficiently directs the optimization toward the global optimum by focusing on the constraints which are violated the most by the current model.

In summary, in order to apply structural SVM to a learning problem, three things are required: a definition of the feature map ψ, the loss function Δ, and an efficient algorithm for the separation oracle. These procedures are all of course highly interdependent and domain-specific. In the next section, we will describe the prevalent approach to solving ranking problems in this setting.

2.2. Ranking with structural SVM

In the case of ranking, the most commonly used feature map is the *partial order* feature (Joachims, 2005):

$$\psi_{po}(q, y) = \sum_{i \in \mathcal{X}_q^+} \sum_{j \in \mathcal{X}_q^-} y_{ij} \left(\frac{\phi(q, i) - \phi(q, j)}{|\mathcal{X}_q^+| \cdot |\mathcal{X}_q^-|} \right), \quad (3)$$

where

$$y_{ij} = \begin{cases} +1 & i \prec_y j \\ -1 & i \succ_y j \end{cases},$$

and $\phi(q, i)$ is a feature map which characterizes the relation between a query q and point i. Intuitively, for each relevant-irrelevant pair (i, j), the difference vector $\phi(q, i) - \phi(q, j)$ is added if $i \prec_y j$ and subtracted otherwise. Essentially, ψ_{po} emphasizes directions in feature space which are in some sense correlated with correct rankings. Since ϕ only depends on the query and a single point, rather than the entire list, it is well-suited for incorporating domain-specific knowledge and features.

Separation oracles have been devised for ψ_{po} in conjunction with a wide variety of ranking evaluation measures (Joachims, 2005; Yue et al., 2007; Chakrabarti et al., 2008), and we give a brief overview in Section 4.

One attractive property of ψ_{po} is that for a fixed w, the ranking y which maximizes $w^\mathsf{T}\psi_{po}(\hat{q}, y)$ is simply $i \in \mathcal{X}$ sorted by descending $w^\mathsf{T}\phi(\hat{q}, i)$. As we will show in the next section, this simple prediction rule can be easily adapted to distance-based ranking.

3. Metric learning to rank

If the query q lies in the same space as the corpus $\mathcal{X}$, a natural ordering is produced by increasing (squared) distance from q: $\|q - i\|^2$. Since our goal is to learn an optimal metric W, distances are computed in the learned space and sorted accordingly: $\|q - i\|_W^2$. This computation is characterized in terms of Frobenius inner products as follows:

$$\begin{aligned} \|q - i\|_W^2 &= (q - i)^\mathsf{T} W (q - i) = \operatorname{tr}\left(W (q - i)(q - i)^\mathsf{T} \right) \\ &= \left\langle W, (q - i)(q - i)^\mathsf{T} \right\rangle_F, \end{aligned}$$

where the second equality follows by the cyclic property of the trace.

This observation suggests a natural choice of ϕ:

$$\phi_M(q, i) \doteq -(q - i)(q - i)^\mathsf{T}. \quad (4)$$

(The change of sign preserves the ordering used in standard structural SVM.) Sorting the corpus by ascending $\|q - i\|_W$ is therefore equivalent to sorting by descending $\langle W, \phi_M(q,i)\rangle_F$. Similarly, by using ϕ_M with ψ_{po}, the ordering y which maximizes the generalized inner product $\langle W, \psi_{po}(q,y)\rangle_F$ is precisely $\mathcal{X}$ in ascending order of distance from q under the metric defined by W.

Thus, by generalizing the vector products in Equations 1 and 2 to Frobenius inner products, we can derive an algorithm to learn a metric optimized for list-wise ranking loss measures.

3.1. Algorithm

Ideally, we would like to solve for the optimal metric W^* which maximizes the margins over all possible rankings for each query. However, since $|\mathcal{Y}|$ is super-exponential in the size of the training set, implementing an exact optimization procedure is not possible with current techniques. Instead, we approximate the full optimization by using a cutting-plane algorithm.

Specifically, our algorithm for learning W is adapted from the 1-Slack margin-rescaling cutting-plane algorithm of Joachims et al. (2009). At a high-level, the algorithm alternates between optimizing the model parameters (in our case, W), and updating the constraint set with a new batch of rankings $(y_1, y_2, \ldots, y_n)$ (one ranking for each point). The algorithm terminates once the empirical loss on the new constraint batch is within a prescribed tolerance $\epsilon > 0$ of the loss on the previous set of constraints.

The key difference between the 1-Slack approach and other similar cutting-plane techniques is that, rather than maintaining a slack variable ξ_q for each $q \in \mathcal{X}$, there is a single slack variable ξ which is shared across all constraint batches, which are in turn aggregated by averaging over each point in the training set.

We introduce two modifications to adapt the original algorithm to metric learning. First, W must be constrained to be positive semi-definite in order to define a valid metric. Second, we replace the standard quadratic regularization $\frac{1}{2}w^\mathsf{T}w$ (or $\frac{1}{2}\operatorname{tr}(W^\mathsf{T}W)$) with $\operatorname{tr}(W)$. Intuitively, this trades an ℓ_2 penalty on the eigenvalues of W for an ℓ_1 penalty, thereby promoting sparsity and low-rank solutions.

The general optimization procedure is listed as Algorithm 1. For compactness, we define

$$\delta\psi_{po}(q, y^*, y) = \psi_{po}(q, y^*) - \psi_{po}(q, y).$$

Algorithm 1 Metric Learning to Rank (MLR).

```
Input: data $\mathcal{X}$, rankings $y_1^*, \ldots, y_n^*$, slack trade-off
    $C > 0$, accuracy threshold $\epsilon > 0$
Output: metric $W \succeq 0$, slack variable $\xi \geq 0$
  $\mathcal{C} \leftarrow \emptyset$
  repeat
    Solve for the optimal metric and slack:
      $(W, \xi) \leftarrow \operatorname{argmin}_{W,\xi} f(W,\xi) = \operatorname{tr}(W) + C\xi$
        s.t. $W \succeq 0$
             $\xi \geq 0$
        $\forall (y_1, y_2, \ldots, y_n) \in \mathcal{C}$ :
          $\frac{1}{n}\sum_{i=1}^{n} \langle W, \delta\psi_{po}(q_i, y_i^*, y_i)\rangle_F \geq$
              $\frac{1}{n}\sum_{i=1}^{n} \Delta(y_i^*, y_i) - \xi$
    for $i = 1$ to $n$ do
      $y_i \leftarrow \operatorname{argmax}_{y\in\mathcal{Y}} \Delta(y_i^*, y) + \langle W, \psi_{po}(q_i, y)\rangle_F$
    end for
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{(y_1, \ldots, y_n)\}$
  until
    $\frac{1}{n}\sum_{i=1}^{n} \Delta(y_i^*, y_i) - \langle W, \delta\psi_{po}(q_i, y_i^*, y_i)\rangle_F \leq \xi + \epsilon$
```

3.2. Implementation

To solve the optimization problem in Algorithm 1, we implemented a gradient descent solver in MATLAB[1]. After each gradient step, the updated W is projected back onto the feasible set of PSD matrices by spectral decomposition.

Although there appears to be a great many feature vectors ($\delta\psi_{po}$) in use in the algorithm, efficient bookkeeping allows us to reduce the overhead of gradient calculations. Note that ξ can be interpreted as the point-wise maximum of a set $\{\xi_1, \xi_2, \ldots\}$, where ξ_i corresponds to the margin constraint for the i^{th} batch. Therefore, at any time when $\xi > 0$, the gradient of the objective $f(W, \xi)$ can be expressed in terms of a single batch $(\hat{y}_1, \ldots, \hat{y}_n)$ which achieves the current largest margin violation:

$$\frac{\partial f}{\partial W} = I - \frac{C}{n}\sum_{i=1}^{n} \delta\psi_{po}(q_i, y_i^*, \hat{y}_i).$$

[1] Source code can be found at `http://www-cse.ucsd.edu/~bmcfee/code/mlr`.

Note that ψ_{po} only appears in Algorithm 1 in the form of averages over constraint batches. This indicates that it suffices to maintain only a single $d \times d$ matrix

$$\Psi = \frac{1}{n} \sum_{i=1}^{n} \delta\psi_{po}(q_i, y_i^*, y_i)$$

for each batch, rather than individual matrices for each point. Because ϕ_M derives from outer-products of the data, each $\psi_{po}(q, y)$ can be factored as

$$\psi_{po}(q, y) = XS(q, y)X^{\mathsf{T}},$$

where the columns of X contain the data, and $S(q, y)$ is a symmetric $n \times n$ matrix with

$$S(q, y) = \sum_{i \in \mathcal{X}_q^+} \sum_{j \in \mathcal{X}_q^-} y_{ij} \frac{(A_{qi} - A_{qj})}{|\mathcal{X}_q^+| \cdot |\mathcal{X}_q^-|}, \qquad (5)$$

$$A_{qx} = -(e_q - e_x)(e_q - e_x)^{\mathsf{T}},$$

and e_i is the i^{th} standard basis vector in $\mathbb{R}^n$. By linearity, this factorization can also be carried through to $\delta\psi_{po}(q, y^*, y)$ and Ψ.

The summation in Equation 5 can be computed more directly by counting the occurrences of A_{qx} with positive and negative sign, and collecting the terms. This can be done in linear time by a single pass through y.

By expressing Ψ in factored form, we can delay all matrix multiplications until the final Ψ computation. Because the $S(q, y)$ can be constructed directly without explicitly building the outer-product matrices A_{qi}, we reduce the number of matrix multiplications at each gradient calculation from $O(n)$ to 2.

4. Ranking measures

Here, we give a brief overview of popular information retrieval evaluation criteria, and how to incorporate them into the learning algorithm.

Recall that the separation oracle (Equation 2) seeks a ranking y which maximizes the sum of the discriminant score $\langle W, \psi_{po}(q, y)\rangle_F$ and the ranking loss $\Delta(y^*, y)$. One property shared by all evaluation criteria under consideration is invariance to permutations within relevant (or irrelevant) sets. As has been previously observed, optimizing over y reduces to finding an optimal interleaving of the relevant and irrelevant sets, each of which has been pre-sorted by the point-wise discriminant score $\langle W, \phi_M(q, i)\rangle_F$ (Yue et al., 2007).

Since all measures discussed here take values in $[0, 1]$ (1 being the score for a perfect ranking), we consider loss functions of the form

$$\Delta(y^*, y) = \text{Score}(y^*) - \text{Score}(y) = 1 - \text{Score}(y).$$

AUC

The area under the ROC curve (AUC) is a commonly used measure which characterizes the trade-off between true positives and false positives as a threshold parameter is varied. In our case, the parameter corresponds to the number of items returned (or, predicted as relevant). AUC can equivalently be calculated by counting the portion of incorrectly ordered pairs (*i.e.*, $j \prec_y i$, i relevant and j irrelevant), and subtracting from 1. This formulation leads to a simple and efficient separation oracle, described by Joachims (2005).

Note that AUC is position-independent: an incorrect pair-wise ordering at the bottom of the list impacts the score just as much as an error at the top of the list. In effect, AUC is a global measure of list-wise cohesion.

Precision-at-k

Precision-at-k (Prec@k) is the fraction of relevant results out of the first k returned. Prec@k is therefore a highly localized evaluation criterion, and captures the quality of rankings for applications where only the first few results matter, *e.g.*, web search.

The separation oracle for Prec@k exploits two facts: there are only $k + 1$ possible values for Prec@k $(0, 1/k, 2/k, \ldots, 1)$, and for any fixed value, the best y is completely determined by the ordering induced by discriminant scores. We can then evaluate all $k+1$ interleavings of the data to find the y which achieves the maximum. See Joachims (2005) for details.

Closely related to Prec@k is the k-nearest neighbor prediction score. In the binary classification setting, the two are related by

$$KNN(q, y; k) = \mathbb{1}\left[\text{Prec@k}(q, y) > 0.5\right],$$

and the Prec@k separation oracle can be easily adapted to k-nearest neighbor. However, in the multiclass setting, the interleaving technique fails because the required fraction of relevant points for correct classification depends not only on the relevance or irrelevance of each point, but the labels themselves.

In informal experiments, we noticed no quantitative differences in performance between metrics trained for (binary) KNN and Prec@k, and we omit KNN from the experiments in Section 5.

Average Precision

Average precision (or Mean Average Precision, MAP) (Baeza-Yates & Ribeiro-Neto, 1999) is the precision-at-k score of a ranking y, averaged over all positions k of relevant documents:

$$AP(q, y) = \frac{1}{|\mathcal{X}_q^+|} \sum_{k=1}^{|\mathcal{X}_q^+|+|\mathcal{X}_q^-|} \text{Prec@k}(y) \mathbb{1}\left[k \in \mathcal{X}_q^+\right].$$

Yue et al. (2007) provides a greedy separation oracle for average precision that runs in time $O(|\mathcal{X}_q^+| \cdot |\mathcal{X}_q^-|)$. Our implementation uses a relatively simpler dynamic programming approach with equivalent asymptotic runtime. (Details are omitted here for brevity.)

Mean Reciprocal Rank

Mean reciprocal rank (MRR) is the inverse position of the first relevant document in y, and is therefore well-suited to applications in which only the first result matters.

Like Prec@k, there is a finite set of possible score values for MRR $(1, 1/2, 1/3, \ldots, 1/(1 + |\mathcal{X}_q^-|))$, and for a fixed MRR score, the optimal y is completely determined. It is similarly straightforward to search over score values for the maximizer. See Chakrabarti et al. (2008) for a more complete treatment of optimizing MRR.

Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) (Järvelin & Kekäläinen, 2000) is similar to MRR, but rather than rewarding only the first relevant document, all of the top k documents are scored at a decaying discount factor. In the present setting with binary relevance levels, the formulation we adopt is expressed as:

$$NDCG(q, y; k) = \frac{\sum_{i=1}^k D(i) \mathbb{1}[i \in \mathcal{X}_q^+]}{\sum_{i=1}^k D(i)}$$

$$D(i) = \begin{cases} 1 & i = 1 \\ 1/\log_2(i) & 2 \leq i \leq k \\ 0 & i > k \end{cases}.$$

Chakrabarti et al. (2008) propose a dynamic programming algorithm for the NDCG separation oracle, which we adapt here.

5. Experiments

To evaluate the MLR algorithm, we performed experiments on both small-scale and large-scale data sets, as described in the next two sections. In all experiments, we fixed the accuracy threshold at $\epsilon = 0.01$.

Table 1. Summary statistics of the UCI data sets: dimensionality, training and test set sizes, and the number of classes. IsoLet's training set was further split into training and validation sets of size 4991 and 1247.

	d	# Train	# Test	# Classes
Balance	4	500	125	3
Ionosphere	34	281	70	2
WDBC	30	456	113	2
Wine	13	143	35	3
IsoLet	170	6238	1559	26

5.1. Classification on UCI data

We first tested the accuracy and dimensionality reduction performance of our algorithm on five data sets from the UCI repository (Asuncion & Newman, 2007): Balance, Ionosphere, WDBC, Wine, and IsoLet. For the first four sets, we generated 50 random 80/20 training and test splits. Each dimension of the data was z-scored by the statistics of the training splits.

For IsoLet, we replicate the experiment of Weinberger et al. (2006) by generating 10 random 80/20 splits of the training set for testing and validation, and then testing on the provided test set. We project by PCA (as computed on the training set) to 170 dimensions, enough to capture 95% of the variance.

Table 1 contains a summary of the data sets used here.

We trained metrics on each data set with the five variants of MLR: MLR-AUC, MLR-Prec@k, MLR-MAP, MLR-MRR, and MLR-NDCG. For comparison purposes, we also trained metrics with Large Margin Nearest Neighbor (LMNN) (Weinberger et al., 2006), Neighborhood Components Analysis (NCA) (Goldberger et al., 2005), and Metric Learning by Collapsing Classes (MLCC) (Globerson & Roweis, 2006).

To evaluate the performance of each algorithm, we tested k-nearest neighbor classification accuracy in the learned metrics. Classification results are presented in Table 2[2]. With the exception of NCA and MLCC on the Balance set, all results on Balance, Ionosphere, WDBC and Wine are within the margin of error. In general, MLR achieves accuracy on par with the best algorithms under comparison, without relying on the input features for selecting target neighbors.

Figure 2 illustrates the dimensionality reduction properties of the MLR algorithms. In all cases, MLR achieves significant reductions in dimensionality from

[2] LMNN accuracy on IsoLet was reported by Weinberger et al. (2006). Dimensionality results were not reported.

Table 2. k-nearest neighbor classification error (%) on learned metrics. Reported error is corresponds to the best choice of C and k.

Algorithm	Bal.	Ion.	Wdbc	Wine	Isolet
MLR-AUC	7.9	12.3	2.7	1.4	4.5
MLR-P@k	8.2	12.3	2.9	1.5	4.5
MLR-MAP	6.9	12.3	2.6	1.0	5.5
MLR-MRR	8.2	12.1	2.6	1.5	4.5
MLR-NDCG	8.2	11.9	2.9	1.6	4.4
LMNN	8.8	11.7	2.4	1.7	4.7
NCA	4.6	11.7	2.6	2.7	10.8
MLCC	5.5	12.6	2.1	1.1	4.4
Euclidean	10.3	15.3	3.1	3.1	8.1

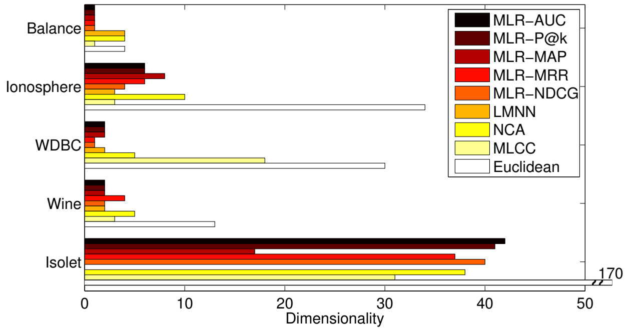

Figure 2. Dimensionality reduction for the UCI data sets. Reported dimensionality is the median number of dimensions necessary to capture 95% of the spectral mass of the best-performing W. "Euclidean" corresponds to the native dimensionality of the data.

Table 3. Summary statistics of eHarmony matching data.

	Matchings	Unique users	Queries
Training	506688	294832	22391
Test	439161	247420	36037

Table 4. Testing accuracy and training time for MLR and SVM-MAP on eHarmony matching data. Time is reported in CPU-seconds, and $|\mathcal{C}|$ is the number of cutting-plane batches before convergence.

Algorithm	AUC	MAP	MRR	Time	$\lvert\mathcal{C}\rvert$
MLR-AUC	0.612	0.445	0.466	232	7
MLR-MAP	0.624	0.453	0.474	2053	23
MLR-MRR	0.616	0.448	0.469	809	17
SVM-MAP	0.614	0.447	0.467	4968	36
Euclidean	0.522	0.394	0.414		

the input space, comparable to the best competing algorithms.

5.2. eHarmony data

To evaluate MLR on a large data set in an information retrieval context, we trained metrics on matching data provided by eHarmony[3]: an online dating service which matches users by personality traits.

For our experiments, we focused on the following simplification of the data and problem: each matching is presented as a pair of users, with a positive label when the match was successful (*i.e.*, users expressed mutual interest), and negative otherwise. Each user is represented by a vector in $\mathbb{R}^{56}$ which describes the user's personality, interests, *etc.* We consider two users mutually relevant if they are presented as a successful match, and irrelevant if the match is unsuccessful. Irrelevance is not assumed for unmatched pairs.

Matchings were collected over two consecutive time intervals of equal length, and split into training (interval 1) and testing (interval 2). The training split contains approximately 295000 unique users, not all of which define useful queries: some appear only in positive matchings, while others appear only in negative matchings. Since these users provide no discriminative data, we omit them from the set of *query users.* Note that such users are still informative, and are included in the training set as results to be ranked.

We further reduce the number of training queries to include only users with at least 2 successful and 5 unsuccessful matchings, leaving approximately 22000 training queries. A summary of the data is presented in Table 3.

We trained metrics with MLR-AUC, MLR-MAP and MLR-MRR. Due to the small number of minimum positive results for each query, we omit MLR-P@k and MLR-NDCG from this experiment. Note that because we are in an information retrieval setting, and not classification, the other metric learning algorithms compared in the previous section do not apply. For comparison, we train models with SVM-MAP (Yue et al., 2007), and feature map $\phi(q, i) = (q - i)$. When training SVM-MAP, we swept over $C \in \{10^{-2}, 10^{-1}, \ldots, 10^{5}\}$.

Table 4 shows the accuracy and timing results for MLR and SVM-MAP. The MLR-MAP and MLR-MRR models show slight, but statistically significant improvement over the SVM-MAP model. Note that the MLR algorithms train in significantly less time

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

than SVM-MAP, and require fewer calls to the separation oracle.

Although MLR improves over baseline Euclidean distance in this retrieval task, it seems that linear models may not suffice to capture complex structure in the data. Generalizing MLR to produce non-linear transformations will be the focus of future research.

6. Conclusion

We have presented a metric learning algorithm which optimizes for ranking-based loss functions. By casting the problem as an information retrieval task, we focus attention on what we believe to be the key quantity of interest: the permutation of data induced by distances.

Acknowledgments

The authors thank Steve Checkoway for helpful suggestions. The authors acknowledge support from NSF Grant DMS-MSPA 0625409 and eHarmony, Inc.

References

Asuncion, A. and Newman, D.J. UCI machine learning repository, 2007. URL http://www.ics.uci.edu/~mlearn/MLRepository.html.

Baeza-Yates, Ricardo A. and Ribeiro-Neto, Berthier. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

Burges, Chris, Shaked, Tal, Renshaw, Erin, Lazier, Ari, Deeds, Matt, Hamilton, Nicole, and Hullender, Greg. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning*, pp. 89–96, New York, NY, USA, 2005. ACM.

Chakrabarti, Soumen, Khanna, Rajiv, Sawant, Uma, and Bhattacharyya, Chiru. Structured learning for non-smooth ranking losses. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 88–96, New York, NY, USA, 2008. ACM.

Crammer, Koby and Singer, Yoram. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2002. ISSN 1532-4435.

Globerson, Amir and Roweis, Sam. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems 18*, pp. 451–458, Cambridge, MA, 2006. MIT Press.

Goldberger, Jacob, Roweis, Sam, Hinton, Geoffrey, and Salakhutdinov, Ruslan. Neighborhood components analysis. In *Advances in Neural Information Processing Systems 17*, pp. 513–520, Cambridge, MA, 2005. MIT Press.

Järvelin, Kalervo and Kekäläinen, Jaana. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 41–48, New York, NY, USA, 2000. ACM.

Joachims, Thorsten. A support vector method for multivariate performance measures. In *Proceedings of the 22nd International Conference on Machine learning*, pp. 377–384, New York, NY, USA, 2005. ACM.

Joachims, Thorsten, Finley, Thomas, and Yu, Chun-Nam John. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009. ISSN 0885-6125.

Tsochantaridis, Ioannis, Joachims, Thorsten, Hofmann, Thomas, and Altun, Yasemin. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6: 1453–1484, 2005. ISSN 1532-4435.

Volkovs, Maksims N. and Zemel, Richard S. Boltzrank: learning to maximize expected ranking gain. In *Proceedings of the 26th annual International Conference on Machine Learning*, pp. 1089–1096, New York, NY, USA, 2009. ACM.

Weinberger, Kilian Q., Blitzer, John, and Saul, Lawrence K. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems 18*, pp. 451–458, Cambridge, MA, 2006. MIT Press.

Xing, Eric P., Ng, Andrew Y., Jordan, Michael I., and Russell, Stuart. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, pp. 505–512, Cambridge, MA, 2003. MIT Press.

Xu, Jun and Li, Hang. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 391–398, New York, NY, USA, 2007. ACM.

Yue, Yisong, Finley, Thomas, Radlinski, Filip, and Joachims, Thorsten. A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 271–278, New York, NY, USA, 2007. ACM.