Distance Metric Learning through Optimization of Ranking

Kreshna Gopal and Thomas R. Ioerger
Department of Computer Science, Texas A&M University
{kgopal, ioerger}@cs.tamu.edu

Abstract

Data preprocessing is important in machine learning, data mining, and pattern recognition. In particular, selecting relevant features in high-dimensional data is often necessary to efficiently construct models that accurately describe the data. For example, many lazy learning algorithms (like k-Nearest Neighbor) rely on feature-based distance metrics to compare input patterns for the purpose of classification or retrieval from a database. In previous work, we introduced Slider, a distance metric learning method that optimizes the weights of features in a protein model-building application (where features are used to describe patterns of electron density around protein macromolecules). In this work, we demonstrate the usefulness of Slider *as a general method for classification, ranking and retrieval, with results on several benchmark datasets. We also compare it to other well-known feature selection or weighting methods.*

1. Introduction

1.1. Distance metric learning, feature weighting, and ranking

Accurate distance measures are critical to many classification, clustering, and retrieval applications. Automated learning of distance metrics is thus widely studied [1], and is closely related to feature selection [2], feature weighting [3], and feature interaction. Learning a distance metric is also related to the problem of ranking [4], especially in learning methods like *k*-Nearest Neighbor and case-based reasoning, where a set of instances are typically ranked based on similarity to a given query instance.

Algorithms for feature selection, learning distance metrics, or learning to rank face numerous challenges when applied to real-world problems. A recent trend is the emergence of applications with large numbers of features (tens or even hundreds of thousands), such as text classification [5], gene expression array analysis, and other genomics applications [6]. In many applications, class labels may not be available [7] – we may instead only know if certain pairs of instances are similar or different.

We argue that learning a distance metric by adjusting the metric proportionately to distances in feature space can lead to data overfitting, especially in complex and noisy domains. Learning a metric from inaccurate and disproportionate feature values can be misleading. A distance value is mostly used in relative comparison among instances, or ranking of instances, based on similarity to a query. This motivates our design for learning a distance metric for the purpose of ranking, through the use of heuristics that try to optimize ranking.

1.2. Feature weighting with Slider

In previous work [8], we presented a distance metric learner called *Slider* for a specific domain (protein structure determination by X-ray crystallography). In this paper, we discuss the effectiveness of Slider as a general distance metric learner. We show empirical results in several benchmark datasets, and compare Slider to other approaches to feature selection and feature weighting.

Like many other feature selection and weighting algorithms, Slider starts with an initial set of weights, iteratively selects and evaluates new weights, retains the best ones, and stops when finding even better weights seems unlikely, or is computationally too expensive. The critical decisions that Slider makes in every iteration are: (1) selecting a new weight vector for evaluation – out of an exponentially large set of options, and (2) evaluating the weight vector to determine if it outperforms the current best one.

Slider uses the following heuristic to evaluate a set of weights: given a training instance, we look at how well the distance metric that uses the weights (like weighted Euclidean) ranks an instance known to be similar to the training instance, relative to a set of

0-7695-3019-2/07 $25.00 © 2007 IEEE
DOI 10.1109/ICDMW.2007.113

known different ones. This is done for a set of training examples, and the average rank of the similar instances is a measure of how good the weight vector is.

Exhaustive search through a space of weights is intractable. Therefore, given true matches and true mismatches for examples, Slider focuses on only those weights that cause each example to be equidistant to its match and its mismatch in Euclidean space. These "crossover" weights are the ones that will influence the accuracy of ranking. Thus, by limiting the space of weights to be searched, and identifying only the weights that are more likely to make a significant difference, the efficiency and effectiveness of learning are largely ensured.

We emphasize the fact that Slider chooses weights that try to maximize the *number of instances* for which true matches are closer to training examples than mismatches in weighted Euclidean space. An alternative approach would be to find weights such the *aggregate distance* between instances and matches is smaller than that between instances and mismatches [1, 7, 9]. The objective function that Slider tries to optimize is an NP-hard *constraint satisfaction problem*, which justifies the use of a heuristic function (based on ranking) to guide the search for optimal weights.

2. Related work

2.1. Feature relevance

A central problem in machine learning is that irrelevant features tend to mask relevant ones, leading to inefficient learning and inaccurate predictions [2]. Potentially useful features are generally defined by an expert (or extracted by automated techniques), and a subset of these features are automatically selected (or highly weighted). Feature selection algorithms are commonly categorized into two major groups: *filters* and *wrappers*. This distinction is based on how feature subsets are evaluated. Filter methods perform the evaluation by using some properties of the features involved, such as correlations, information gain, dependencies, separability, etc. [10]. Wrapper methods use part of the data sample to iteratively evaluate subsets of selected features by running the induction program itself, based on techniques such as cross-validation [11]. The advantage of wrapper approaches is that features are selected based on the bias of the induction algorithm. The disadvantage of wrappers is their high computational cost.

The feature selection problem can also be thought of as a *heuristic search* over a space of states, each of which represents a subset of features [2]. For example, in *forward selection*, we start with an empty feature set and iteratively add features; in *backward elimination*, we start with all features, and then remove one feature at a time. The organization of the search is also critical, since an exhaustive search is intractable. Many feature selection problems have been shown to be NP-hard [12].

2.2. Distance metric learning

Another approach to determine feature relevance is to apply a weighting function to features, which effectively assigns *degrees* of relevance to features [3]. Most feature weighting methods employ some variation of *gradient descent*, such as the perceptron update rule, least-mean squares [13], and neural network [14].

Fully weighted distance metrics are also often used, such as the Mahalanobis distance, or one that maximizes the ratio of intra-class variance to inter-class variance [1]. A number of algorithms have been proposed to learn a Mahalanobis distance e.g. Xing *et al*. [7] adopt a convex optimization method based on semidefinite programming for clustering. Shalev-Shwartz *et al*. [15] propose a method to learn a distance metric by defining a threshold for pairwise distances between similar examples, and one for different examples, and uses a loss function to induce the difference between the thresholds. Goldberger *et al*. [16] uses *neighborhood component analysis* and gradient descent to minimize the probability of Nearest Neighbor classification. Other methods proposed include quadratic programming [9] and energy-based models [17].

Distance metric learning can be *global* or *local*. In the former, the learning aims at satisfying the constraints of *all* labeled training data. In the latter, constraints are satisfied only in the neighborhood of the problem instance in feature space. Short and Fukunaga [18] use Euclidean distance to first define a neighborhood of the query, and then creates a new local distance based on statistics of instances within the neighborhood. Other methods proposed for local distance metric learning include adapting the shape of neighborhoods around query points to capture local feature relevance [19] and local Linear Discriminant Analysis [1].

2.3. Ranking

Learning a weighted distance metric for the purpose of *classification* has been widely studied and successfully applied. Nonetheless, we often wish to retrieve instances, such as relevant documents from the Web, or potentially useful planning solutions from a

plan library. In these applications, there is no explicit classification, and similarity is usually measured by a continuous metric. The objective is to retrieve and rank rather than classify.

In RankNet [20], gradient descent and a probabilistic cost function are used to rank search results from the Web. Joachims [21] uses a support vector machine approach for learning retrieval functions of search engines. Cohen *et al*. [4] propose other methods to combine multiple preference functions to create an ordering, and apply them for Web searches.

3. Methods

3.1. A two-phase case retrieval strategy

Consider a case-based reasoning system with a database $\mathbb{D}$ consisting of N cases, and a set of query instances Q, each of which is represented by a set F of numeric features. We define the distance between a query q and case x in $\mathbb{D}$ by the weighted Euclidean metric:

$$d_F(q,x) = \sqrt{\sum_{i=1}^{|F|} w_i (q_i - x_i)^2} \qquad (1)$$

where x_i and q_i are the i^{th} feature of x and q respectively, and w_i is the weight of that feature. We compute $d_F(q,x)$ for all x's in $\mathbb{D}$ and rank the x's according to their distances to q in increasing order. Our aim in the proposed filtering method is to rank as many truly similar cases as possible in the top k ranked instances, where $k \ll N$ (e.g. $k/N \approx 0.01$). A more accurate (and typically expensive) method can then be used rank the k cases. If the features are properly selected (or weighted), it will enrich the top k cases with more matches, and increase the expected probability of retrieving true matches. It will also imply affordance of a smaller k, and hence more efficient retrieval.

3.2. The Slider algorithm

We now describe in details how Slider weights features. First we consider two-component mixtures (i.e. involving two features, where their weights sum up to 1) and then extend it to an arbitrary number of features. The distance metric we use is weighted Euclidean – nonetheless this method can be applied to Minkowski distances in general.

The weighted Euclidean distance between two instances x and y, using two features i and j (with weights w_i and w_j respectively, where $w_i + w_j = 1$) is defined as:

$$d_{\{i,j\}}(x,y) = \sqrt{w_i (x_i - y_i)^2 + w_j (x_j - y_j)^2} \qquad (2)$$

We can drop the square root in (2), since it is a monotonic transformation i.e.

$$\begin{aligned} d_{\{i,j\}}(x,y) &= w_i (x_i - y_i)^2 + w_j (x_j - y_j)^2 \\ &= (1-w)(x_i - y_i)^2 + w(x_j - y_j)^2 \end{aligned} \qquad (3)$$

where w is set to w_j, the weight of feature j. Consider an instance x that has y as its closest neighbor according to feature f_i, and z as its closest neighbor according to feature f_j. As w "slides" from 0 to 1, there is a weight w_c at which $d_{\{i,j\}}(x,y) = d_{\{i,j\}}(x,z)$. This point is called a "crossover". By expanding $d_{\{i,j\}}(x,y) = d_{\{i,j\}}(x,z)$, we get:

$$(1-w)(x_i - y_i)^2 + w(x_j - y_j)^2 = (1-w)(x_i - z_i)^2 + w(x_j - z_j)^2 \qquad (4)$$

Solving for w, and setting it to w_c, we get:

$$w_c = \frac{(x_i - z_i)^2 - (x_i - y_i)^2}{(x_j - y_j)^2 - (x_i - y_i) + (x_i - z_i)^2 - (x_j - z_j)^2} \qquad (5)$$

In other words, w_c is a weight where there is a net increase (or decrease) in accuracy of classification (or ranking), depending on which of y and z is truly closer to x (determined by an independent, objective metric). When there is an increase in accuracy (i.e. the match is closer to x than the mismatch, for all weights above w_c), it is referred to as "positive crossover", and "negative crossover" otherwise. We can find the crossover weights for a training set of 3-tuples, and choose the optimum weight w^* that represents the best compromise between positive and negative crossovers.

Crossover weights can also be determined by considering two subsets of features (instead of just two features). Consider two feature subsets A and B, with corresponding Euclidean distances d_A and d_B respectively. A composite metric, d_{A+B}, can be defined as $d_{A+B}(x,y) = wd_A(x,y) + (1 - w)d_B(x,y)$. As w is increased from 0 to 1, it may cause a switch of neighbors, as described earlier. Thus, w can be used to determine the new weight vector that increases accuracy, based on crossover points. In Slider, we randomly choose one feature (singleton set A) and evaluate it against all remaining features (set B).

Slider starts by assigning the same weight to all features. It then uses a hill-climbing approach by iteratively choosing a feature (randomly), adjusting its weight to make the distance metric more accurate, and stopping when there is no net increase in accuracy (in terms of ranking).

After a weight vector is chosen (by using crossover weights), it is evaluated based on how well the corresponding weighted Euclidean distance ranks matches relative to mismatches. We use an independent validation set V for this evaluation. For each instance in

V, we find a true match from a database $\mathbb{D}$. Our goal is to estimate the average rank of the match against any sample of mismatches drawn from $\mathbb{D}$. Given a weight vector $\vec{w}$ that we want to evaluate, we define the rank $R(v,\vec{w})$ for an instance *v* in *V* as the average rank over a set $\mathbf{\Phi}$ of *n* sets of mismatching instances from $\mathbb{D}$. Let $\mathbf{\Phi} = \{\mathbf{\Phi_1}, \mathbf{\Phi_2}, \ldots, \mathbf{\Phi_n}\}$. $R(v,\vec{w})$ is defined as:

$$R(v,\vec{w}) = \frac{1}{|\Phi|}\sum_{i=1}^{n} rank(v,\Phi_i,\vec{w}) \quad (6)$$

where $rank(v,\mathbf{\Phi_i},\vec{w})$ is the rank of the match of validation instance *v* (relative to all instances in $\mathbf{\Phi_i}$), using the underlying metric with weight vector $\vec{w}$. Note that lower rank implies more similar to the query instance (i.e. the match should ideally have rank = 1). For practical purposes, we estimate *R* using a randomly drawn sample of 100 mismatches (singleton set $\mathbf{\Phi}$) from our database. Then we compute the average $\overline{R}$ over all instances *v* in *V* i.e.

$$\overline{R}(V,\vec{w}) = \frac{1}{|V|}\sum_{v\in V} R(v,\vec{w}) \quad (7)$$

The pseudo-code of Slider is given in Figure 1.

Inputs:
1. Training set *T* of ‹instance, match, mismatch› 3-tuples;
2. Validation set *V* – for each instance in *V*, 1 match & 100 mismatches;
3. Set *F* of features.

Output: Optimized weight vector $\vec{w}^*$ = ‹w_1*, w_2*, …, $w_{|F|}$*›

Initialize weights uniformly i.e. $w_i = 1/|F|$, $1 \le i \le |F|$.
repeat
  Select feature *f* randomly.
  Set w_f to 0 and adjust other weights proportionally so that they add up to 1.
  Find all crossover points from *T* by sliding w_f from 0 to 1.
  Find "optimum" weight of *f*, w_f*.
  Set w_f to w_f* and other weights are proportionally decreased to keep sum of weights 1.
  Compute mean rank of matches for new weight vector, $\overline{R}(V,\vec{w})$ using (7).
  if $\overline{R}(V,\vec{w})$ decreases (improves) *then*
    best weights $\vec{w}^*$ is set to $\vec{w}$.
until $\overline{R}(V,\vec{w})$ reaches a plateau *or* number of iterations exceeds a threshold.
return $\vec{w}^*$

Figure 1. The Slider algorithm.

4. Empirical results

4.1. Bioinformatics application

In this section we present empirical results on Slider from the protein crystallography domain, based on a system that interprets electron density maps to determine the 3D structures of proteins [8]. We use this domain to compare Slider to other feature selection and weighting methods.

The protein model-building program searches a database of about 50,000 spherical regions of electron density patterns to find matches to help interpret regions in a new electron density map. Given an electron density pattern for a spherical region in an unknown map, 400 putatively similar patterns are pre-selected from the database, using a computationally efficient weighted distance metric, based on 76 rotation-invariant features that characterize the local density (the features used are described in [8]). These filtered cases are then re-evaluated by a more expensive *density correlation* measure to determine truly matching structural fragments.

Figure 2 illustrates how the weighted Euclidean metric (with weights determined by Slider) performs compared to the non-weighted Euclidean distance in the retrieval of matches in the top $k = 400$ cases (from the database of about 50,000 cases). We compare the two metrics for various tolerances in defining similarity, using density correlation. (Density correlation ranges from 0 to 1; if the absolute best match has a correlation of 0.90, then with a tolerance of 0.01, all cases with correlation between 0.89 and 0.90 are considered to be similar – all others are different.) With feature weights determined by Slider, about twice as many matches are found at each level of tolerance (as opposed to uniform weights). These trends are also observed for other feature-based similarity metrics, such as Manhattan or Minkowski distance of order 3.

In Figure 3, we show that Slider outperforms several other standard feature selection and weighting methods. In all the algorithms we use the mean rank of matches [$\overline{R}$, as defined in (7)] to tune the weights. Figure 3 shows the retrieval accuracy for the weighted Euclidean distance metric using Slider weights, uniform weights (i.e. non-weighted), and weights determined by the following algorithms:

(1) DIET [22] is a wrapper approach that searches a space of p+1 discrete possible weights: 0, 1/p, 2/p, …, (p -1)/p, 1 (the results shown are based on $p = 10$). The operators in this heuristic search replace the current weight of a feature by either the next larger or smaller

value in the allowed set based on improvement in ranking.

(2) Sequential forward and backward selection (SFS and SBS) start from an empty and full set of features respectively, and greedily add or remove one feature at a time.

(3) A Linear Programming (LP) approach that tries to optimize the aggregate difference in the (squared) distance values between instance and their matches and mismatches. We used GLPK (GNU Linear Programming Kit) to solve the optimization problem.

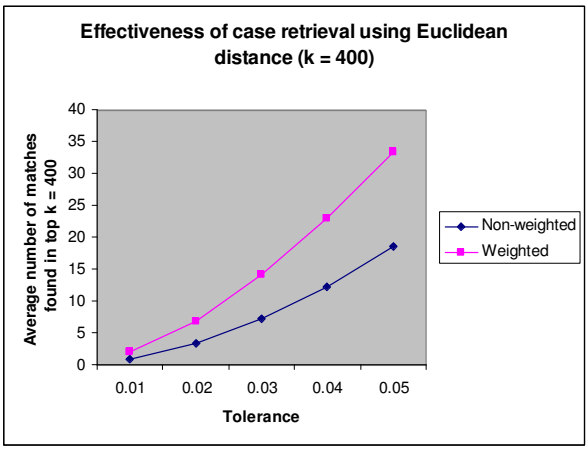

Figure 2. Weighted Euclidean distance places more matches in the top 400 cases than non-weighted Euclidean distance (the weights are determined by Slider) in the protein crystallography domain. This is true for various values of tolerance, which determines how lenient we are in defining similarity – with higher tolerance, more cases will qualify as being similar.

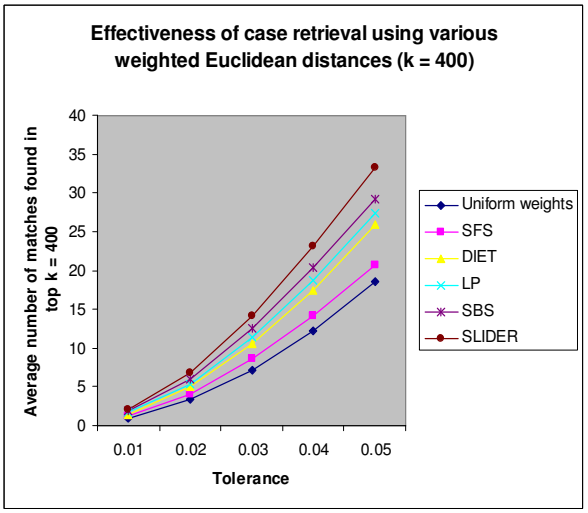

Figure 3. Slider is more effective in weighting features (such that true matches are highly ranked and retrieved) as compared to other algorithms.

4.2. Slider for classification: UCI datasets

In this section, we present empirical results on how Slider performs for classification, using the following datasets from the UCI repository (www.ics.uci.edu/~mlearn/MLRepository.html): Wine, Ionosphere, Isolet, Pima Indian Diabetes, Iris, and Letter Recognition. In Table 1, we compare non-weighted Euclidean distance to the metric that Slider learns, based on three criteria (the results are for 10 test sets, each with 100 examples): (1) the accuracy of 1-Nearest Neighbor classification; (2) the rank of a match relative to 10 mismatches (i.e. $\overline{R}$), averaged over the test set; (3) the "density" of matches in the top 10% of the database i.e. instances closest to the query. This is compared to the number of matches per instance for the entire database. The results for this criterion show how the distance metrics "enrich" the 10% closest neighbors with matches.

We observe that the weighted Euclidean distance (with weights determined by Slider) outperforms Euclidean distance with uniform weights in terms of all the three criteria mentioned above. (For simplicity we report results on 1-NN and rank relative to 10 instances. Nonetheless, similar trends are observed for larger neighborhoods and more mismatches for ranking.) Even in cases where the improvement in nearest neighbor classification is not significant (e.g. Iris and Letter Recognition), we observe that the improvement based on the latter two measures is still significant.

5. Conclusion

In this paper, we argue that distance metrics should be tuned so that they are effective in properly *ranking* similar patterns relative to different ones. To this end, we propose Slider, an algorithm both *selects* and *evaluates* feature weights based on ranking. We showed how Slider performs on benchmark datasets. It improves nearest neighbor classification, compared to Euclidean distance with uniform weights. Furthermore, Slider ranks matches better relative to mismatches, and "enriches" the neighborhoods of query instances with more matches.

We also show the effectiveness of Slider in a complex, noisy real-world bioinformatics application, where we compared Slider to other feature selection algorithms. The distance metric learning and case retrieval methods we propose are potentially useful in other domains, especially those with high-dimensional and noisy data, expensive case matching, large databases, and the objective to rank instances (like Web pages), or retrieve as many good instances from a database.

Table 2. Comparing Euclidean distance with uniform weights to the metric learned by Slider, based on three criteria: (1) the percentage accuracy of 1-NN classification; (2) $\overline{R}$, the rank of a match relative to 10 mismatches; (3) the number of matches in the top 10% instances. We also show the ratio of the total number of matches to the size of the database (in the "Entire database" column). The results are for 10 test sets (each with 100 examples).

Dataset	1-NN		$\overline{R}$		No. of matches per instance		
	Euclidean	Slider	**Euclidean**	Slider	**Entire database**	**Top 10% Euclidean**	**Top 10%** Slider
WINE	78 ± 4.4	96 ± 1.7	3.4 ± .2	1.7 ±. 2	.34 ± .01	.62 ± .02	.88 ± .02
IONOSPHERE	87 ± 1.7	92 ± 1.9	4.7 ± .3	3.9 ± .4	.54 ± .01	.73 ± .02	.79 ± .03
ISOLET	84 ± 3.2	87 ± 4.2	1.8 ± .1	1.7 ± .2	.04 ± .00	.27 ± .01	.29 ± .01
PIMA INDIAN DIA.	67 ± 4.4	70 ± 4.8	5.6 ± .2	5.2 ± .2	.54 ± .02	.63 ± .02	.66 ± .01
IRIS	96 ± 2.2	95 ± 2.3	1.8 ± .1	1.5 ± .2	.33 ± .00	.87 ± .01	.89 ± .01
LETTER REC.	95 ± 1.3	96 ± 1.8	3.7 ± .3	3.3 ± .3	.04 ± .00	.14 ± .01	.17 ± .01

6. References

[1] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification and regression," *Advances in Neural Information Processing Systems*, 8, pp. 409-415, 1996.

[2] A.L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, pp. 245-271, 1997.

[3] N. Littlestone, "Learning quickly when irrelevant attributes abound: a new linear threshold algorithm," *Machine Learning*, vol. 8, pp. 293-321, 1992.

[4] W.W. Cohen, R.E. Schapire, and Y. Singer, "Learning to order things," *Advances in Neural Processing Systems*, vol. 10, Denver: MIT Press, 1997.

[5] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of Machine Learning Research*, 3, pp. 1289-1305, 2003.

[6] M. Berens, H. Liu, and L. Yu, "Fostering biological relevance in feature selection for microarray data," *IEEE Intelligent Systems*, 20(6), pp. 71-73, 2005.

[7] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-information," *Advances in Neural Processing Systems*, 15, pp. 505-512, 2003.

[8] K. Gopal, T.D. Romo, J.C. Sacchettini, and T.R. Ioerger, "Determining relevant features to recognize electron density patterns in X-ray protein crystallography," *Journal of Bioinformatics & Computational Biology*, 3(3), pp. 645-676, 2005.

[9] W. Tsang and J.T. Kwok, "Distance metric learning with kernels," *Proc. of the International Conference on Artificial Neural Networks*, pp. 126-129, 2003.

[10] K. Kira and L.A. Rendell, "A practical approach to feature selection," *Proc. of the 9th International Conference on Machine Learning*, pp. 249-256, 1992.

[11] G. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," *Proc. of the 11th International Conference on Machine Learning*, pp. 121-129, 1994.

[12] A.L. Blum and R.L. Rivest, "Training a 3-node neural networks in NP-complete," *Neural Networks*, 5, pp. 117-127, 1992.

[13] B. Widrow and M.E. Hoff, "Adaptive switching circuits," *Proc. of the 3rd Annual Workshop on Computational Learning Theory*, pp. 371-383, 1990.

[14] S. Baluja and D. Pomerleau, "Dynamic relevance: vision-based focus of attention using artificial neural networks," *Artificial Intelligence*, vol. 97, pp. 381-395, 1997.

[15] S. Shalev-Shwartz, Y. Singer, and A.Y. Ng, "Online and batch learning of pseudo-metrics," *Proc. of the 21st International Conference on Machine Learning*, pp. 94, 2004.

[16] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov, "Neighbourhood Component Analysis," *Neural Information Processing Systems*," 17, pp. 513-520, 2004.

[17] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminately, with application to face recognition," *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition,* pp. 539-546, 2005.

[18] R. Short and K. Fukunaga, "A new nearest neighbor distance measure," *Proc. of the 5th IEEE International Conference Pattern Recognition*, pp. 81-86, 1980.

[19] C. Domeniconi, J. Peng, and D. Gunopulos, "An adaptive metric machine for pattern classification," *Advances in Neural Processing Systems*, 13, pp. 451-457, 2002.

[20] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," *Proc. of the 22nd International Conference on Machine Learning,* pp. 89-96, 2005.

[21] T. Joachims, "Optimizing search engines using clickthrough data," *Proc. of the 8th ACM Conference on Knowledge Discovery and Data Mining,* pp. 133-142, 2002.

[22] R. Kohavi, P. Langley, and Y. Yun, "The utility of feature weighting in nearest-neighbor algorithms," *Lecture Notes in Computer Science*, 1224, pp. 85-92, 1997.