Metric Learning

ICML 2010 Tutorial

Brian Kulis
University of California at Berkeley

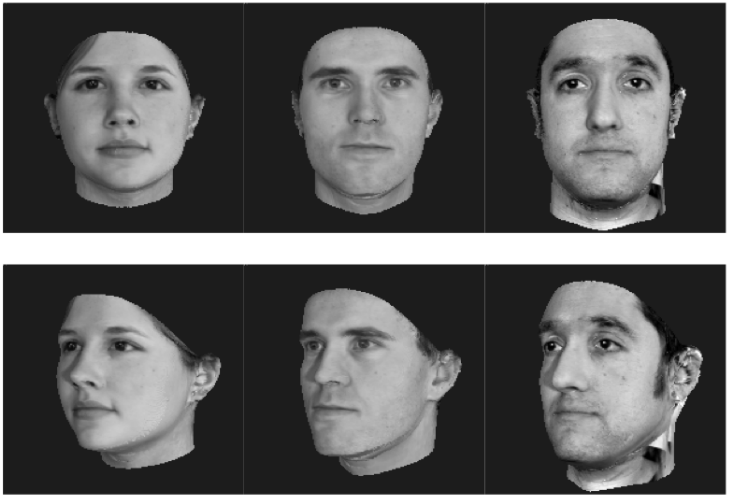
June 21, 2010

Introduction

Learning problems with distances and similarities

- *k*-means
- Support vector machines
- *k*-nearest neighbors
- Most algorithms that employ kernel methods
- Other clustering algorithms (agglomerative, spectral, etc)
- ...

Choosing a distance function

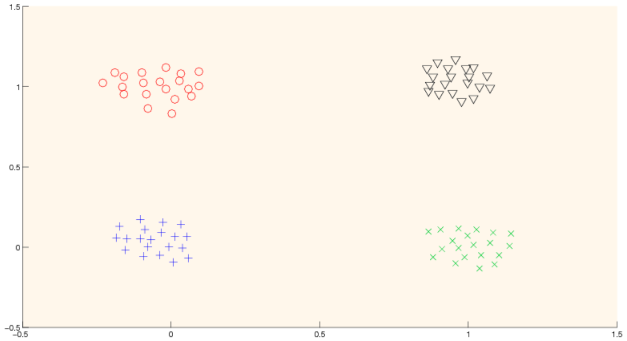


Choosing a distance function

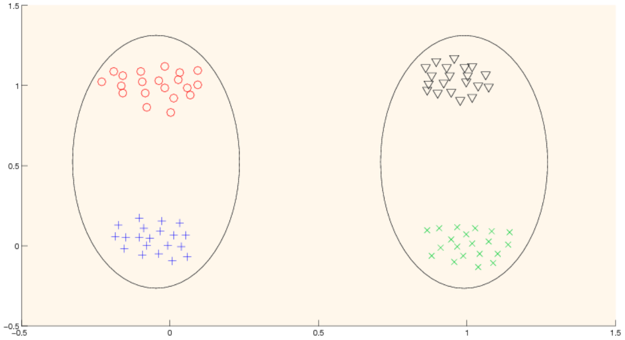
Example: UCI Wine data set

- 13 features
  - $9/13$ features have mean value in $[0, 10]$
  - $3/13$ features have mean value in $[10, 100]$
  - One feature has a mean value of 747 (with std 315)
- Using a standard distance such as Euclidean distance, the largest feature dominates the computation
  - That feature may not be important for classification
- Need a weighting of the features that improves classification or other tasks

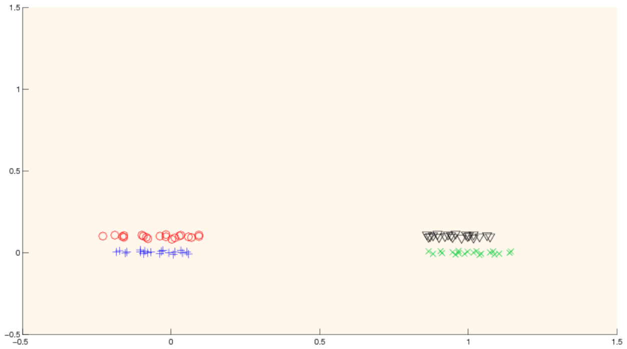
Example: Four Blobs



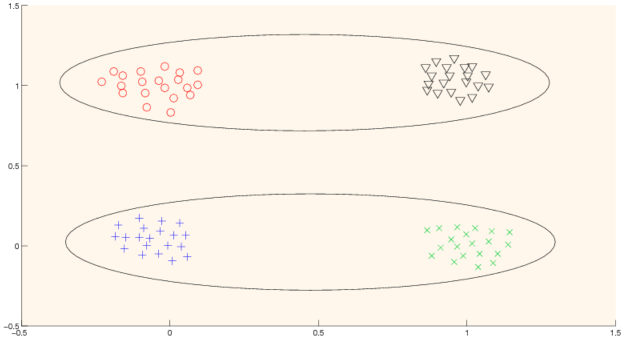
Example: Four Blobs



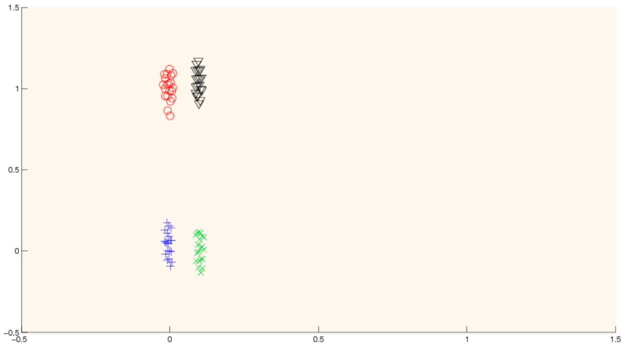
Example: Four Blobs



Example: Four Blobs



Example: Four Blobs



Metric Learning as Learning Transformations

- Feature re-weighting
  - Learn weightings over the features, then use standard distance (e.g., Euclidean) after re-weighting
  - Diagonal Mahalanobis methods (e.g., Schultz and Joachims)
  - Number of parameters grows linearly with the dimensionality d
- Full linear transformation
  - In addition to scaling of features, also rotates the data
  - For transformations from d dimensions to d dimensions, number of parameters grows quadratically in d
  - For transformations to $r < d$ dimensions, this is linear dimensionality reduction
- Non-linear transformation
  - Variety of methods
  - Neural nets
  - Kernelization of linear transformations
  - Complexity varies from method to method

Supervised vs Unsupervised Metric Learning

- Unsupervised Metric Learning
  - Dimensionality reduction techniques
  - Principal Components Analysis
  - Kernel PCA
  - Multidimensional Scaling
  - In general, not the topic of this tutorial...
- Supervised and Semi-supervised Metric Learning
  - Constraints or labels given to the algorithm
  - Example: set of similarity and dissimilarity constraints
  - This is the focus of the tutorial

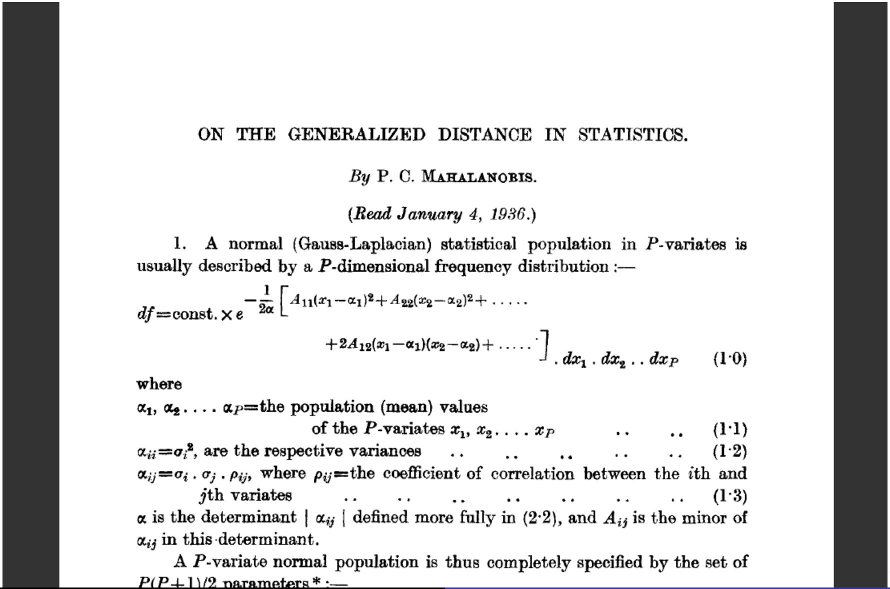
Themes of the tutorial

- Not just a list of algorithms
  - General principles
  - Focus on a few key methods
- Recurring ideas
  - Scalability
  - Linear vs non-linear
  - Online vs offline
  - Optimization techniques utilized
  - Statements about general formulations
- Applications
  - Where is metric learning applied?
  - Success stories
  - Limitations

Outline of Tutorial

- Motivation
- Linear metric learning methods
    - Mahalanobis metric learning
    - Per-example methods
- Non-linear metric learning methods
    - Kernelization of Mahalanobis methods
    - Other non-linear methods
- Applications
- Conclusions

Mahalanobis Distances



ON THE GENERALIZED DISTANCE IN STATISTICS.

By P. C. MAHALANOBIS.

(*Read January 4, 1936.*)

1. A normal (Gauss-Laplacian) statistical population in P-variates is usually described by a P-dimensional frequency distribution :—

$$df = \text{const.} \times e^{-\frac{1}{2\alpha}\left[A_{11}(x_1-\alpha_1)^2 + A_{22}(x_2-\alpha_2)^2 + \ldots\ldots + 2A_{12}(x_1-\alpha_1)(x_2-\alpha_2) + \ldots\ldots\right]} . dx_1 . dx_2 \,.\,. dx_P \qquad (1\cdot 0)$$

where

$\alpha_1, \alpha_2 \ldots . \alpha_P$ = the population (mean) values of the P-variates $x_1, x_2 \ldots . x_P$ .. .. (1·1)

$\alpha_{ii} = \sigma_i^2$, are the respective variances .. .. (1·2)

$\alpha_{ij} = \sigma_i . \sigma_j . \rho_{ij}$, where ρ_{ij} = the coefficient of correlation between the ith and jth variates .. .. (1·3)

α is the determinant $|\alpha_{ij}|$ defined more fully in (2·2), and A_{ij} is the minor of α_{ij} in this determinant.

A P-variate normal population is thus completely specified by the set of $P(P+1)/2$ parameters* :—

Mahalanobis Distances

- Assume the data is represented as N vectors of length d: $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$
- Squared Euclidean distance

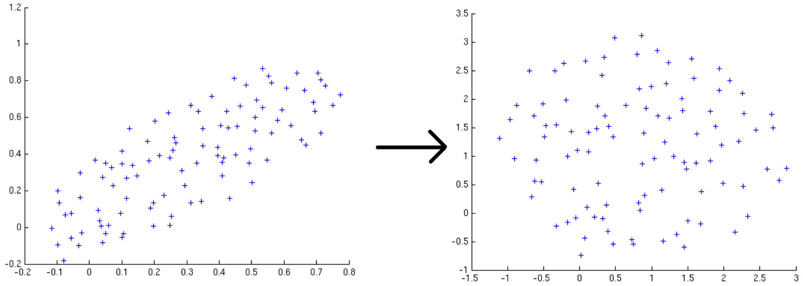
$$\begin{aligned} d(\mathbf{x}_1, \mathbf{x}_2) &= \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 \\ &= (\mathbf{x}_1 - \mathbf{x}_2)^T(\mathbf{x}_1 - \mathbf{x}_2) \end{aligned}$$

- Let $\Sigma = \sum_{i,j}(\mathbf{x}_i - \mu)(\mathbf{x}_j - \mu)^T$
- The "original" Mahalanobis distance:

$$d_M(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 - \mathbf{x}_2)^T \Sigma^{-1} (\mathbf{x}_1 - \mathbf{x}_2)$$

Mahalanobis Distances

- Equivalent to applying a *whitening transform*



Mahalanobis Distances

- Assume the data is represented as N vectors of length d: $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$
- Squared Euclidean distance

$$\begin{aligned} d(\mathbf{x}_1, \mathbf{x}_2) &= \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 \\ &= (\mathbf{x}_1 - \mathbf{x}_2)^T(\mathbf{x}_1 - \mathbf{x}_2) \end{aligned}$$

- Mahalanobis distances for metric learning
  - Distance parametrized by $d \times d$ positive semi-definite matrix A:

$$d_A(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 - \mathbf{x}_2)^T A(\mathbf{x}_1 - \mathbf{x}_2)$$

- Used for many existing metric learning algorithms

[Xing, Ng, Jordan, and Russell; NIPS 2002]
[Bar-Hillel, Hertz, Shental, and Weinshall; ICML 2003]
[Bilenko, Basu, and Mooney; ICML 2004]
[Globerson and Roweis; NIPS 2005]
[Weinberger, Blitzer, and Saul; NIPS 2006]

Mahalanobis Distances

$$d_A(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 - \mathbf{x}_2)^T A(\mathbf{x}_1 - \mathbf{x}_2)$$

- Why is A positive semi-definite (PSD)?
  - If A is not PSD, then d_A could be negative
  - Suppose $\mathbf{v} = \mathbf{x}_1 - \mathbf{x}_2$ is an eigenvector corresponding to a negative eigenvalue λ of A

$$\begin{aligned} d_A(\mathbf{x}_1, \mathbf{x}_2) &= (\mathbf{x}_1 - \mathbf{x}_2)^T A(\mathbf{x}_1 - \mathbf{x}_2) \\ &= \mathbf{v}^T A\mathbf{v} \\ &= \lambda\mathbf{v}^T\mathbf{v} = \lambda < 0 \end{aligned}$$

Mahalanobis Distances

- Properties of a metric:
  - $d(\mathbf{x}, \mathbf{y}) \geq 0$
  - $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$
  - $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$
  - $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$
- d_A is not technically a metric
  - Analogous to Euclidean distance, need the square root:

$$\sqrt{d_A(\mathbf{x}_1, \mathbf{x}_2)} = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T A(\mathbf{x}_1 - \mathbf{x}_2)}$$

- Square root of the Mahalanobis distance satisfies all properties if A is strictly positive definite, but if A is positive semi-definite then second property is not satisfied
  - Called a *pseudo-metric*
- In practice, most algorithms work only with d_A

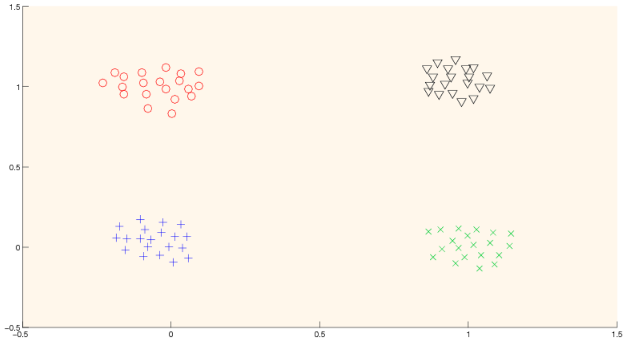
Mahalanobis Distances

- Can view d_A as the squared Euclidean distance after applying a linear transformation
  - Decompose $A = G^T G$ via Cholesky decomposition
  - (Alternatively, take eigenvector decomposition $A = V \Lambda V^T$ and look at $A = (\Lambda^{1/2} V^T)^T (\Lambda^{1/2} V^T)$)
- Then we have

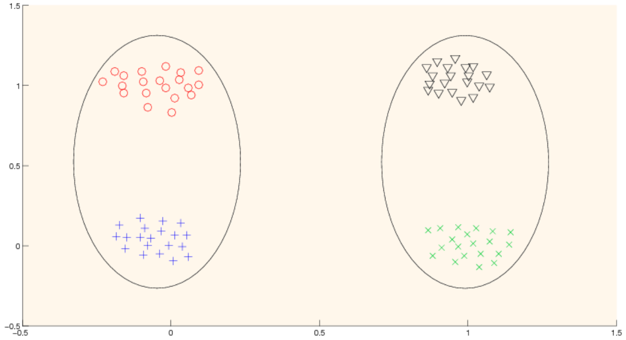
$$\begin{aligned} d_A(\mathbf{x}_1, \mathbf{x}_2) &= (\mathbf{x}_1 - \mathbf{x}_2)^T A (\mathbf{x}_1 - \mathbf{x}_2) \\ &= (\mathbf{x}_1 - \mathbf{x}_2) G^T G (\mathbf{x}_1 - \mathbf{x}_2) \\ &= (G\mathbf{x}_1 - G\mathbf{x}_2)^T (G\mathbf{x}_1 - G\mathbf{x}_2) \\ &= \|G\mathbf{x}_1 - G\mathbf{x}_2\|_2^2 \end{aligned}$$

- Mahalanobis distance is just the squared Euclidean distance after applying the linear transformation G

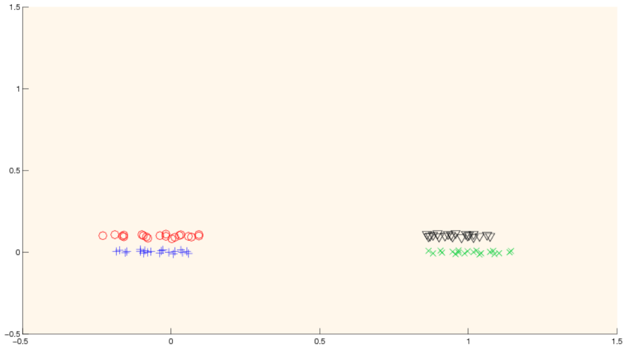
Example: Four Blobs



Example: Four Blobs



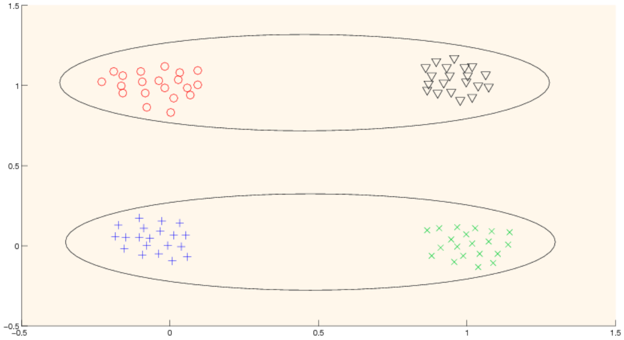
Example: Four Blobs



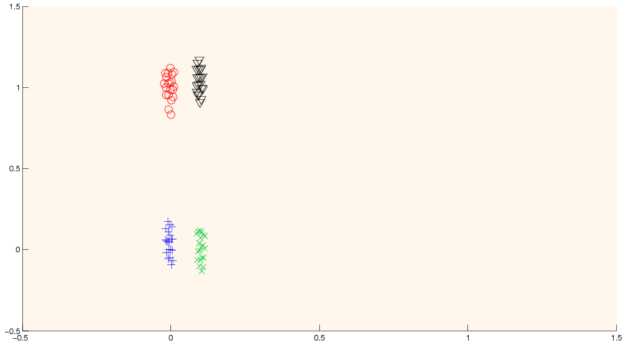
- Want to learn:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \qquad G = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{\epsilon} \end{pmatrix}$$

Example: Four Blobs



Example: Four Blobs

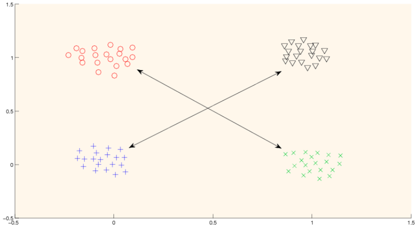


- Want to learn:

$$A = \begin{pmatrix} \epsilon & 0 \\ 0 & 1 \end{pmatrix} \qquad G = \begin{pmatrix} \sqrt{\epsilon} & 0 \\ 0 & 1 \end{pmatrix}$$

Drawbacks to Mahalanobis Metric Learning

- Memory overhead grows quadratically with the dimensionality of the data
    - Does not scale to high-dimensional data ($d = O(10^6)$ for many image embeddings)
- Only works for linearly separable data



- **Cannot seemingly be applied to "real" data!**
- These drawbacks will be discussed later

Metric Learning Problem Formulation

- Typically 2 main pieces to a Mahalanobis metric learning problem
    - A set of constraints on the distance
    - A regularizer on the distance / objective function
- In the constrained case, a general problem may look like:

$$\begin{array}{ll} \min_A & r(A) \\ \text{s.t.} & c_i(A) \leq 0 \quad 0 \leq i \leq C \\ & A \succeq 0 \end{array}$$

- r is a regularizer/objective on A and c_i are the constraints on A
- An unconstrained version may look like:

$$\min_{A \succeq 0} r(A) + \lambda \sum_{i=1}^{C} c_i(A)$$

Defining Constraints

- Similarity / Dissimilarity constraints
  - Given a set of pairs $\mathcal{S}$ of points that should be similar, and a set of pairs of points $\mathcal{D}$ of points that should be dissimilar
  - A single constraint would be of the form
    $$d_A(\mathbf{x}_i, \mathbf{x}_j) \leq \ell$$
    for $(i, j) \in \mathcal{S}$ or
    $$d_A(\mathbf{x}_i, \mathbf{x}_j) \geq u$$
    for $(i, j) \in \mathcal{D}$
  - Easy to specify given class labels
- Relative distance constraints
  - Given a triple $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ such that the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ should be smaller than the distance between $\mathbf{x}_i$ and $\mathbf{x}_k$, a single constraint is of the form
    $$d_A(\mathbf{x}_i, \mathbf{x}_j) \leq d_A(\mathbf{x}_i, \mathbf{x}_k) - m,$$
    where m is the margin
  - Popular for ranking problems

Defining Constraints

- Aggregate distance constraints
  - Constrain the sum of all pairs of same-class distances to be small, e.g.,

$$\sum_{ij} y_{ij} d_A(\mathbf{x}_i, \mathbf{x}_j) \leq 1$$

  where $y_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are in the same class, and 0 otherwise
- Other constraints
  - Non-parametric probability estimation constraints
  - Constraints on the generalized inner product $\mathbf{x}_i^T A \mathbf{x}_j$:

$$d_A(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T A \mathbf{x}_i + \mathbf{x}_j^T A \mathbf{x}_j - 2\mathbf{x}_i^T A \mathbf{x}_j$$

Defining the Regularizer or Objective

- Loss/divergence functions
  - Squared Frobenius norm: $\|A - A_0\|_F^2$
  - LogDet divergence: $\mathrm{tr}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d$
  - General loss functions $D(A, A_0)$
  - Will discuss several of these later
- Other regularizers
  - $\|A\|_F^2$
  - $\mathrm{tr}(AC_0)$ (i.e., if C_0 is the identity, this is the trace norm)

Choosing a Regularizer

- Depends on the problem!
- Example 1: $\mathrm{tr}(A)$
  - Trace function is the sum of the eigenvalues
  - Analogous to the ℓ_1 penalty, promotes sparsity
  - Leads to low-rank A
- Example 2: LogDet Divergence
  - Defined only over positive semi-definite matrices
  - Makes computation simpler
  - Possesses other desirable properties
- Example 3: $\|A\|_F^2$
  - Arises in many formulations
  - Easy to analyze and optimize

Defining the Optimization

- Many existing Mahalanobis distance learning methods can be obtained simply by choosing a regularizer/objective and constraints
- We will discuss properties of several of these

Xing et al.'s MMC

Problem posed as follows:

$$\begin{aligned} \max_A \quad & \textstyle\sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{D}} \sqrt{d_A(\mathbf{x}_i,\mathbf{x}_j)} \\ \text{s.t.} \quad & c(A) = \textstyle\sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{S}} d_A(\mathbf{x}_i,\mathbf{x}_j) \leq 1 \\ & A \succeq 0. \end{aligned}$$

- Here, $\mathcal{D}$ is a set of pairs of dissimilar pairs, $\mathcal{S}$ is a set of similar pairs
- Objective tries to maximize sum of dissimilar distances
- Constraint keeps sum of similar distances small
    - Use square root in regularizer to avoid trivial solution

[Xing, Ng, Jordan, and Russell; NIPS 2002]

Xing et al.'s MMC

Algorithm

- Based on gradient descent over the objective followed by an iterative projection step to find a feasible A
  - Constraint $c(A)$ is linear in A, can be solved cheaply
  - Orthogonal projection onto $A \succeq 0$ achieved by setting A's negative eigenvalues to 0
  - Iterative between these two steps to find feasible A for both constraints, then take a step in the gradient of the objective
- Despite relative simplicity, the algorithm is fairly slow (many eigenvalue decompositions required)
- Does not scale to large problems
- Objective and constraints only look at the sums of distances

Schultz and Joachims

Problem formulated as follows:

$$\begin{aligned} \min_{A} \quad & \|A\|_F^2 \\ \text{s.t.} \quad & d_A(\mathbf{x}_i, \mathbf{x}_k) - d_A(\mathbf{x}_i, \mathbf{x}_j) \geq 1 \;\; \forall (i,j,k) \in \mathcal{R} \\ & A \succeq 0. \end{aligned}$$

- Constraints in $\mathcal{R}$ are relative distance constraints
- There may be no solution to this problem; introduce slack variables

$$\begin{aligned} \min_{A,\xi} \quad & \|A\|_F^2 + \gamma \textstyle\sum_{(i,j,k) \in \mathcal{R}} \xi_{ijk} \\ \text{s.t.} \quad & d_A(\mathbf{x}_i, \mathbf{x}_k) - d_A(\mathbf{x}_i, \mathbf{x}_j) \geq 1 - \xi_{ijk} \;\; \forall (i,j,k) \in \mathcal{R} \\ & \xi_{ijk} \geq 0 \;\; \forall (i,j,k) \in \mathcal{R} \\ & A \succeq 0. \end{aligned}$$

[Schultz and Joachims; NIPS 2002]

Schultz and Joachims

Algorithm

- Key simplifying assumption made
    - $A = M^T DM$, where M is assumed fixed and known and D is diagonal

$$\begin{aligned} d_A(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i - \mathbf{x}_j)^T A(\mathbf{x}_i - \mathbf{x}_j) \\ &= (\mathbf{x}_i - \mathbf{x}_j)^T M^T DM(\mathbf{x}_i - \mathbf{x}_j) \\ &= (M\mathbf{x}_i - M\mathbf{x}_j)^T D(M\mathbf{x}_i - M\mathbf{x}_j) \end{aligned}$$

    - Effectively constraining the optimization to diagonal matrices
    - Resulting optimization problem is very similar to SVMs, and resulting algorithm is similar
- By choosing M to be a matrix of data points, method can be kernelized
- Fast algorithm, but less general than full Mahalanobis methods

Kwok and Tsang

Problem formulated as follows:

$$\begin{aligned}
\min_{A,\xi,\gamma} \quad & \|A\|_F^2 + \tfrac{C_S}{N_S} \textstyle\sum_{(\mathbf{x}_i,\mathbf{x}_j)\in S} \xi_{ij} + \tfrac{C_D}{N_D} \textstyle\sum_{(\mathbf{x}_i,\mathbf{x}_j)\in D} \xi_{ij} - C_D\gamma\nu \\
\text{s.t.} \quad & d_I(\mathbf{x}_i,\mathbf{x}_j) \geq d_A(\mathbf{x}_i,\mathbf{x}_j) - \xi_{ij} \quad \forall (\mathbf{x}_i,\mathbf{x}_j) \in S \\
& d_A(\mathbf{x}_i,\mathbf{x}_j) - d_I(\mathbf{x}_i,\mathbf{x}_j) \geq \gamma - \xi_{ij} \quad \forall (\mathbf{x}_i,\mathbf{x}_j) \in D \\
& \xi_{ij} \geq 0 \\
& \gamma \geq 0 \\
& A \succeq 0.
\end{aligned}$$

- Same regularization as Schultz and Joachims
- Similarity/dissimilarity constraints instead of relative distance constraints
- No simplifying assumptions made about A

[Kwok and Tsang; ICML 2003]

Neighbourhood Components Analysis

Problem formulated as follows:

$$\begin{array}{ll} \max_{A} & \sum_i \sum_{j \in C_i, j \neq i} \frac{\exp(-d_A(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k \neq i} \exp(-d_A(\mathbf{x}_i, \mathbf{x}_k))} \\ \text{s.t.} & A \succeq 0. \end{array}$$

- C_i is the set of points in the same class as point $\mathbf{x}_i$ (not including $\mathbf{x}_i$)

Motivation

- Minimize the leave-one-out KNN classification error
    - LOO error function is discontinuous
    - Replace by a softmax; each point $\mathbf{x}_i$ chooses a nearest neighbor $\mathbf{x}_j$ based on probability

$$p_{ij} = \frac{\exp(-d_A(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k \neq i} \exp(-d_A(\mathbf{x}_i, \mathbf{x}_k))}$$

[Goldberger, Roweis, Hinton, and Salakhutdinov; NIPS 2004]

Neighbourhood Components Analysis

Algorithm

- Problem is *non-convex*
- Rewrite in terms of G, where $A = G^T G$
  - Eliminates $A \succeq 0$ constraint
- Run gradient descent over G

Properties

- Easy to control the rank of A: just optimize over low-rank G
- Simple, unconstrained optimization
- No guarantee of global solution

MCML

Recall NCA probabilities

$$p_{ij} = \frac{\exp(-d_A(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k \neq i} \exp(-d_A(\mathbf{x}_i, \mathbf{x}_k))}$$

- Introduce an “ideal” probability distribution p^0_{ij}:

$$p^0_{ij} \propto \begin{cases} 1 & \text{if } i \text{ and } j \text{ from same class} \\ 0 & \text{otherwise.} \end{cases}$$

- Minimize divergence between p^0 and p:

$$\begin{array}{ll} \min\limits_{A} & KL(p^0, p) \\ \text{s.t.} & A \succeq 0. \end{array}$$

[Globerson and Roweis; NIPS 2005]

MCML

Properties

- Unlike NCA, MCML is *convex*
- Global optimization possible
  - Algorithm based on optimization over the dual
  - Similar to Xing: gradient step plus projection
  - Not discussed in detail in this tutorial

A Closer Look at Some Algorithms

- As can be seen, several objectives are possible
- We will take a look at 3 algorithms in-depth for their properties
- POLA
  - An *online* algorithm for learning metrics with provable regret
  - Also the first supervised metric learning algorithm that was shown to be kernelizable
- LMNN
  - Very popular method
  - Algorithm scalable to billions of constraints
  - Extensions for learning multiple metrics
- ITML
  - Objective with several desirable properties
  - Simpler kernelization construction
  - Online variant

POLA

Setup

- Estimate d_A in an online manner
- Also estimate a threshold b
- At each step t, observe tuple $(\mathbf{x}_t, \mathbf{x}'_t, y_t)$
  - $y_t = 1$ if $\mathbf{x}_t$ and $\mathbf{x}'_t$ should be similar; -1 otherwise
- Consider the following loss

$$\ell_t(A, b) = \max(0, y_t(d_A(\mathbf{x}_t, \mathbf{x}'_t)^2 - b) + 1)$$

  - Hinge loss, margin interpretation
- Appropriately update A and b each iteration

[Shalev-Shwartz, Singer, and Ng; NIPS 2004]

Online Learning Setup

- Define the total loss to be

$$L = \sum_{t=1}^{T} \ell_t(A_t, b_t)$$

- Online learning methods compare the loss to the best *fixed*, offline predictor A^* and threshold b^*
- Define *regret* for T total timesteps as

$$R_T = \sum_{t=1}^{T} \ell_t(A_t, b_t) - \sum_{t=1}^{T} \ell_t(A^*, b^*)$$

- Design an algorithm that minimizes the regret
  - Same setup as in other online algorithms (classification, regression)
  - Modern optimization methods achieve $O(\sqrt{T})$ regret for a general class, and $O(\log T)$ for some special cases

POLA Algorithm

- Consider the following convex sets

$$\begin{aligned} C_t &= \{(A,b) \mid \ell_t(A,b) = 0\} \\ C_a &= \{(A,b) \mid A \succeq 0, b \geq 1\} \end{aligned}$$

- Consider orthogonal projections:

$$P_C(\mathbf{x}) = \mathrm{argmin}_{\mathbf{y} \in C} \|\mathbf{x} - \mathbf{y}\|_2^2$$

- Think of (A, b) as a vector in $d^2 + 1$ dimensional space
- Each step, project onto C_t, then project onto C_a

POLA Algorithm

- Projection onto C_t
  - Let $\mathbf{v}_t = \mathbf{x}_t - \mathbf{x}'_t$
  - Then projection is computed in closed form as

$$\begin{aligned}\alpha_t &= \frac{\ell_t(A_t, b_t)}{\|\mathbf{v}_t\|_2^4 + 1}\\ \hat{A}_t &= A_t - y_t \alpha_t \mathbf{v}_t \mathbf{v}_t^T\\ \hat{b}_t &= b_t + \alpha_t y_t\end{aligned}$$

- Projection onto C_a
  - Orthogonal projection onto positive semi-definite cone obtained by setting negative eigenvalues to 0
  - But, update from A_t to $\hat{A}_t$ was *rank-one*
  - Only 1 negative eigenvalue (interlacing theorem)
  - Thus, only need to compute the smallest eigenvalue and eigenvector (via a power method or related) and subtract it off

Analysis of POLA

- **Theorem:** Let $(\mathbf{x}_1, \mathbf{x}'_1, y_1), ..., (\mathbf{x}_T, \mathbf{x}'_T, y_T)$ be a sequence of examples and let R be such that $\forall t, R \geq \|\mathbf{x}_t - \mathbf{x}'_t\|_2^4 + 1$. Assume there exists an $A^* \succeq 0$ and $b^* \geq 1$ such that $\ell_t(A^*, b^*) = 0 \ \forall t$. Then the following bound holds for all $T \geq 1$:

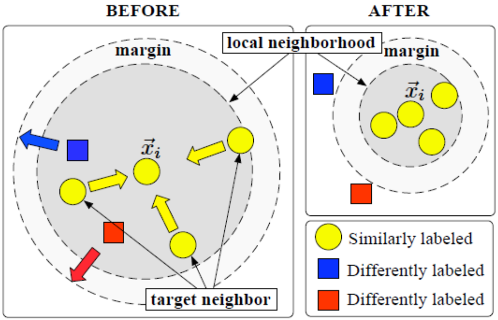
$$\sum_{t=1}^{T} \ell_t(A_t, b_t)^2 \leq R(\|A^*\|_F^2 + (b^* - b_1)^2).$$

  - Note that since $\ell_t(A^*, b^*) = 0$ for all t, the total loss is equal to the regret
  - Can generalize this to a regret bound in the case when $\ell_t(A^*, b^*)$ does not always equal 0
- Can also run POLA in batch settings

POLA in Context

- Can we think of POLA in the framework presented earlier?
  - Yes—regularizer is $\|A\|_F^2$ and constraints are defined by the hinge loss ℓ_t
  - Similar to both Schultz and Joachims, and Kwok and Tsang
- We will see later that POLA can also be kernelized to learn non-linear transformations
- In practice, POLA does not appear to be competitive with current state-of-the-art

LMNN



- Similarly to Schultz and Joachims, utilize relative distance constraints
  - A constraint $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{R}$ has the property that $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors of the same class, and $\mathbf{x}_i$ and $\mathbf{x}_k$ are of different classes

[Weinberger, Blitzer, and Saul; NIPS 2005]

LMNN

- Problem Formulation
  - Also define set $\mathcal{S}$ of pairs of points $(\mathbf{x}_i, \mathbf{x}_j)$ such that $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors in the same class
  - Want to minimize sum of distances of pairs of points in $\mathcal{S}$
  - Also want to satisfy the relative distance constraints
- Mathematically:

$$\begin{array}{cc} \min\limits_{A} & \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{S}} d_A(\mathbf{x}_i, \mathbf{x}_j) \\ \text{s.t.} & d_A(\mathbf{x}_i, \mathbf{x}_k) - d_A(\mathbf{x}_i, \mathbf{x}_j) \geq 1 \;\; \forall (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{R} \\ & A \succeq 0. \end{array}$$

LMNN

- Problem Formulation
  - Also define set $\mathcal{S}$ of pairs of points $(\mathbf{x}_i, \mathbf{x}_j)$ such that $\mathbf{x}_i$ and $\mathbf{x}_j$ are neighbors in the same class
  - Want to minimize sum of distances of pairs of points in $\mathcal{S}$
  - Also want to satisfy the relative distance constraints
- Mathematically:

$$\begin{aligned} \min_{A,\xi} \quad & \textstyle\sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{S}} d_A(\mathbf{x}_i,\mathbf{x}_j) + \gamma \sum_{(\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k)\in\mathcal{R}} \xi_{ijk} \\ \text{s.t.} \quad & d_A(\mathbf{x}_i,\mathbf{x}_k) - d_A(\mathbf{x}_i,\mathbf{x}_j) \geq 1 - \xi_{ijk} \quad \forall(\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k)\in\mathcal{R} \\ & A \succeq 0, \xi_{ijl} \geq 0. \end{aligned}$$

  - Introduce slack variables

Comments on LMNN

- Algorithm
  - Special-purpose solver
  - Relies on subgradient computations
  - Ignores inactive constraints
  - Example: MNIST—3.2 billion constraints in 4 hours
  - Software available
- Performance
  - One of the best-performing methods
  - Works in a variety of settings

LMNN Extensions

Learning with Multiple Local Metrics

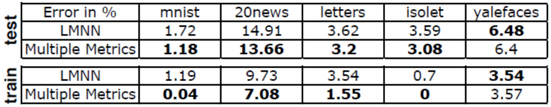
- Learn several local Mahlanobis metrics instead a single global one
  - Cluster the training data into k partitions
  - Denote c_i as the corresponding cluster for $\mathbf{x}_i$
  - Learn k Mahalanobis distances $A_1, ..., A_k$
- Formulation

$$\begin{aligned} \min_{A} \quad & \textstyle\sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{S}} d_{A_{c_j}}(\mathbf{x}_i,\mathbf{x}_j) \\ \text{s.t.} \quad & d_{A_{c_k}}(\mathbf{x}_i,\mathbf{x}_k) - d_{A_{c_j}}(\mathbf{x}_i,\mathbf{x}_j) \geq 1 \ \ \forall(\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_k)\in\mathcal{R} \\ & A_i \succeq 0 \ \ \forall i. \end{aligned}$$

  - Introduce slack variables as with standard LMNN

[Weinberger and Saul; ICML 2008]

LMNN Results



	Error in %	mnist	20news	letters	isolet	yalefaces
test	LMNN	1.72	14.91	3.62	3.59	**6.48**
	Multiple Metrics	**1.18**	**13.66**	**3.2**	**3.08**	6.4
train	LMNN	1.19	9.73	3.54	0.7	**3.54**
	Multiple Metrics	**0.04**	**7.08**	**1.55**	**0**	3.57

- Results show improvements using multiple metrics
- Weinberger and Saul also extend LMNN to use ball trees for fast search
  - No time to go into details, see paper

ITML and the LogDet Divergence

- We take the regularizer to be the Log-Determinant Divergence:

$$D_{\ell d}(A, A_0) = \mathrm{trace}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d$$

- Problem formulation:

$$\begin{array}{lll} \min_A & D_{\ell d}(A, A_0) & \\ \text{s.t.} & (\mathbf{x}_i - \mathbf{x}_j)^T A(\mathbf{x}_i - \mathbf{x}_j) \leq u & \text{if } (i,j) \in \mathcal{S} \text{ [similarity constraints]} \\ & (\mathbf{x}_i - \mathbf{x}_j)^T A(\mathbf{x}_i - \mathbf{x}_j) \geq \ell & \text{if } (i,j) \in \mathcal{D} \text{ [dissimilarity constraints]} \end{array}$$

[Davis, Kulis, Jain, Sra, and Dhillon; ICML 2007]

LogDet Divergence: Properties

$$D_{\ell d}(A, A_0) = \operatorname{trace}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d,$$

- Properties:
  - Scale-invariance

    $$D_{\ell d}(A, A_0) = D_{\ell d}(\alpha A, \alpha A_0), \quad \alpha > 0$$

  - In fact, for any invertible M

    $$D_{\ell d}(A, A_0) = D_{\ell d}(M^T A M, M^T A_0 M)$$

  - Expansion in terms of eigenvalues and eigenvectors ($A = V \Lambda V^T, A_0 = U \Theta U^T$):

    $$D_{\ell d}(A, A_0) = \sum_{i,j} (\mathbf{v}_i^T \mathbf{u}_j)^2 \left(\frac{\lambda_i}{\theta_j} - \log \frac{\lambda_i}{\theta_j} \right) - d$$

Existing Uses of LogDet

- Information Theory
  - Differential relative entropy between two same-mean multivariate Gaussians equal to LogDet divergence between covariance matrices
- Statistics
  - LogDet divergence is known as Stein's loss in the statistics community
- Optimization
  - BFGS update can be written as:

$$\begin{aligned} \min_{B} \quad & D_{\ell d}(B, B_t) \\ \text{subject to} \quad & B\, s_t = y_t \quad (\text{“Secant Equation”}) \end{aligned}$$

  - $s_t = x_{t+1} - x_t,\ y_t = \nabla f_{t+1} - \nabla f_t$

Key Advantages

- Simple algorithm, easy to implement in Matlab
- Method can be kernelized
- Scales to millions of data points
- Scales to high-dimensional data (text, images, etc.)
- Can incorporate locality-sensitive hashing for *sub-linear time* similarity searches

The Metric Learning Problem

$$D_{\ell d}(A, A_0) = \mathrm{trace}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d$$

- **ITML Goal:**

$$\begin{array}{lll} \min_A & D_{\ell d}(A, A_0) & \\ \text{s.t.} & (\mathbf{x}_i - \mathbf{x}_j)^T A(\mathbf{x}_i - \mathbf{x}_j) \leq u & \text{if } (i,j) \in \mathcal{S} \text{ [similarity constraints]} \\ & (\mathbf{x}_i - \mathbf{x}_j)^T A(\mathbf{x}_i - \mathbf{x}_j) \geq \ell & \text{if } (i,j) \in \mathcal{D} \text{ [dissimilarity constraints]} \end{array}$$

Algorithm: Successive Projections

- Algorithm: project successively onto each linear constraint — **converges to globally optimal solution**
- Use projections to update the Mahalanobis matrix:

$$\begin{array}{ll} \min\limits_{A} & D_{\ell d}(A, A_t) \\ \text{s.t.} & (\mathbf{x}_i - \mathbf{x}_j)^T A (\mathbf{x}_i - \mathbf{x}_j) \leq u \end{array}$$

- Can be solved by $O(d^2)$ rank-one update:

$$A_{t+1} = A_t + \beta_t A_t (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T A_t$$

- Advantages:
  - Automatic enforcement of positive semidefiniteness
  - Simple, closed-form projections
  - No eigenvector calculation
  - Easy to incorporate slack for each constraint

Recent work in Mahalanobis methods

- Recent work has looked at other regularizers, such as $\text{tr}(A)$, which learns low-rank matrices
- Improvements in online metric learning (tighter bounds)
- Kernelization for non-linear metric learning, the topic of the next section

LEGO

- Online bounds proven for a variant of POLA based on LogDet regularization
  - Combines the best of both worlds
- Minimize the following function at each timestep

$$f_t(A) = D_{\ell d}(A, A_t) + \eta_t \ell_t(A, \mathbf{x}_t, \mathbf{y}_t)$$

  - A_t is the current Mahalanobis matrix
  - η_t is the learning rate
  - $\ell_t(A, \mathbf{x}_t, \mathbf{y}_t)$ is a loss function, e.g.

$$\ell_t(A, \mathbf{x}_t, \mathbf{y}_t) = \frac{1}{2}(d_A(\mathbf{x}_t, \mathbf{y}_t) - p)^2$$

- For appropriate choice of step size, can guarantee $O(\sqrt{T})$ regret
- Empirically outperforms POLA significantly in practice

[Jain, Kulis, Dhillon, and Grauman; NIPS 2009]

Non-Mahalanobis methods: Local distance functions

- General approach
  - Learn a distance function for *every* training data point
  - Given m features per point, denote d_m^{ij} as the distance between the m-th feature in points $\mathbf{x}_i$ and $\mathbf{x}_j$
  - Denote w_m^j as a weight for feature m of point $\mathbf{x}_j$
  - Then the distance between an arbitary (e.g., test) image $\mathbf{x}_i$ and a training image $\mathbf{x}_j$ is

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{m=1}^{M} w_m^j d_m^{ij}$$

- At test time
  - Given test image $\mathbf{x}_i$, compute $d(\mathbf{x}_i, \mathbf{x}_j)$ between $\mathbf{x}_i$ and every training point $\mathbf{x}_j$
  - Sort distances to find nearest neighbors

[Frome, Singer, Sha, and Malik; ICCV 2007]

Non-Mahalanobis methods: Local distance functions

Optimization framework

- Denote $\mathbf{w}_j$ as the vector of weights w_m^j
- As before, construct triples (i, j, k) of points such that the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ should be smaller than the distance between $\mathbf{x}_i$ and $\mathbf{x}_k$
- Formulate the following problem:

$$\begin{aligned} \min_W \quad & \textstyle\sum_j \|\mathbf{w}_j\|_2^2 \\ \text{s.t.} \quad & d(\mathbf{x}_i, \mathbf{x}_k) - d(\mathbf{x}_i, \mathbf{x}_j) \geq 1 \;\; \forall (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{R} \\ & \mathbf{w}_j \geq 0 \;\; \forall j. \end{aligned}$$

Non-Mahalanobis methods: Local distance functions

Optimization framework

- Denote $\mathbf{w}_j$ as the vector of weights w_m^j
- As before, construct triples (i, j, k) of points such that the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ should be smaller than the distance between $\mathbf{x}_i$ and $\mathbf{x}_k$
- Formulate the following problem:

$$\begin{array}{rc} \min\limits_{W} & \sum_j \|\mathbf{w}_j\|_2^2 + \gamma \sum_{(i,j,k)} \xi_{ijk} \\ \text{s.t.} & d(\mathbf{x}_i, \mathbf{x}_k) - d(\mathbf{x}_i, \mathbf{x}_j) \geq 1 - \xi_{ijk} \quad \forall (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{R} \\ & \mathbf{w}_j \geq 0 \quad \forall j. \end{array}$$

  - Introduce slack variables as before
  - Very similar to LMNN and other relative distance methods!

Non-Mahalanobis methods: Local distance functions

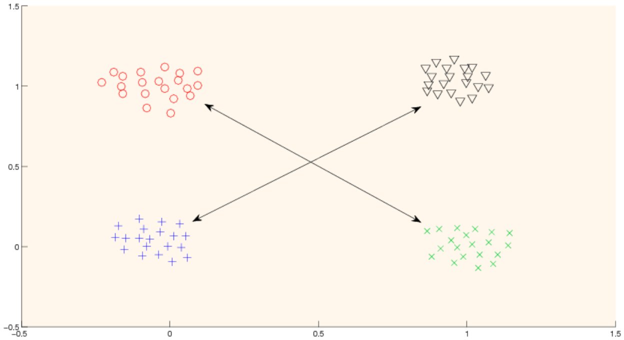
- Schultz and Joachims

$$\begin{aligned}\min_{A} \quad & \|A\|_F^2 \\ \text{s.t.} \quad & d_A(\mathbf{x}_i, \mathbf{x}_k) - d_A(\mathbf{x}_i, \mathbf{x}_j) \geq 1 \;\; \forall (i,j,k) \in \mathcal{R} \\ & A \succeq 0.\end{aligned}$$

- Frome et al.

$$\begin{aligned}\min_{W} \quad & \textstyle\sum_j \|\mathbf{w}_j\|_2^2 \\ \text{s.t.} \quad & d(\mathbf{x}_i, \mathbf{x}_k) - d(\mathbf{x}_i, \mathbf{x}_j) \geq 1 \;\; \forall (\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) \in \mathcal{R} \\ & \mathbf{w}_j \geq 0 \;\; \forall j.\end{aligned}$$

Linear Separability



- No linear transformation for this grouping

Kernel Methods

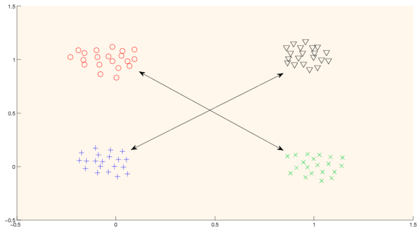
- Map input data to higher-dimensional "feature" space:

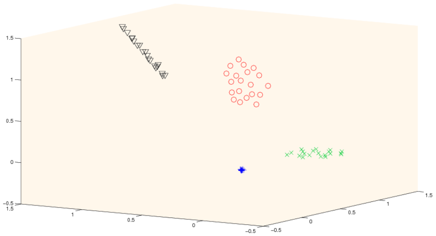
$$\mathbf{x} \rightarrow \varphi(\mathbf{x})$$

- Idea: Run machine learning algorithm in feature space
- Use the following mapping:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

Mapping to Feature Space





Kernel Methods

- Map input data to higher-dimensional "feature" space:

$$\mathbf{x} \rightarrow \varphi(\mathbf{x})$$

- Idea: Run machine learning algorithm in feature space

- Use the following mapping:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

- Kernel function: $\kappa(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle$

- "Kernel trick" — no need to explicitly form high-dimensional features

- In this example: $\langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle = (\mathbf{x}^T\mathbf{y})^2$

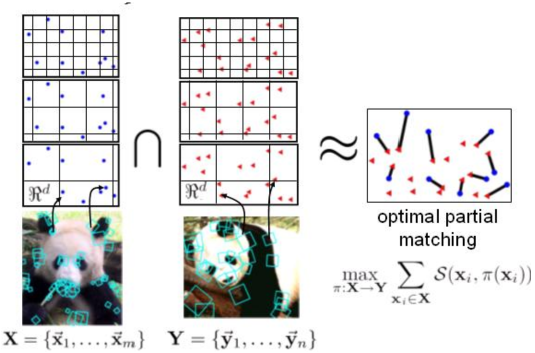
Kernel Methods: Short Intro

- Main idea
  - Take an existing learning algorithm
  - Write it using inner products
  - Replace inner products $\mathbf{x}^T\mathbf{y}$ with kernel functions $\varphi(\mathbf{x})^T\varphi(\mathbf{y})$
  - If $\varphi(\mathbf{x})$ is a non-linear function, then algorithm has been *implicitly* non-linearly mapped
- Examples of kernel functions

$$\begin{aligned}
\kappa(\mathbf{x},\mathbf{y}) &= (\mathbf{x}^T\mathbf{y})^p \quad \text{Polynomial Kernel} \\
\kappa(\mathbf{x},\mathbf{y}) &= \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|_2^2}{2\sigma^2}\right) \quad \text{Gaussian Kernel} \\
\kappa(\mathbf{x},\mathbf{y}) &= \tanh(c(\mathbf{x}^T\mathbf{y})+\theta) \quad \text{Sigmoid Kernel}
\end{aligned}$$

- Kernel functions also defined over objects such as images, trees, graphs, etc.

Example: Pyramid Match Kernel


$\Re^d$
$\Re^d$
$\cap$
$\approx$
optimal partial matching
$\max_{\pi:\mathbf{X}\to\mathbf{Y}} \sum_{\mathbf{x}_i\in\mathbf{X}} S(\mathbf{x}_i, \pi(\mathbf{x}_i))$
$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}$
$\mathbf{Y} = \{\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_n\}$

- Compute *local* image features
- Perform an approximate matching between features of two images
- Use multi-resolution histograms
- View as a dot product between high-dimensional vectors

[Grauman and Darrell, ICCV 2005]

Example: *k*-means

Recall the *k*-means clustering algorithm

- Repeat until convergence:
  - Compute the means of every cluster π_c

$$\mu_c = \frac{1}{|\pi_c|} \sum_{\mathbf{x}_i \in \pi_c} \mathbf{x}_i$$

  - Reassign points to their closest mean by computing

$$\|\mathbf{x} - \mu_c\|_2^2$$

  for every data point $\mathbf{x}$ and every cluster π_c

Kernelization of *k*-means

- Expand $\|\mathbf{x} - \mu_c\|_2^2$ as

$$\mathbf{x}^T\mathbf{x} - \frac{2\sum_{\mathbf{x}_i \in \pi_c} \mathbf{x}^T\mathbf{x}_i}{|\pi_c|} + \frac{\sum_{\mathbf{x}_i, \mathbf{x}_j \in \pi_c} \mathbf{x}_i^T\mathbf{x}_j}{|\pi_c|^2}$$

- No need to explicitly compute the mean; just compute this for every point to every cluster

Example: *k*-means

Recall the *k*-means clustering algorithm

- Repeat until convergence:
    - Compute the means of every cluster π_c

$$\mu_c = \frac{1}{|\pi_c|} \sum_{\mathbf{x}_i \in \pi_c} \mathbf{x}_i$$

    - Reassign points to their closest mean by computing

$$\|\mathbf{x} - \mu_c\|_2^2$$

  for every data point $\mathbf{x}$ and every cluster π_c

Kernelization of *k*-means

- Expand $\|\mathbf{x} - \mu_c\|_2^2$ as

$$\kappa(\mathbf{x}, \mathbf{x}) - \frac{2 \sum_{\mathbf{x}_i \in \pi_c} \kappa(\mathbf{x}, \mathbf{x}_i)}{|\pi_c|} + \frac{\sum_{\mathbf{x}_i, \mathbf{x}_j \in \pi_c} \kappa(\mathbf{x}_i, \mathbf{x}_j)}{|\pi_c|^2}$$

- Replace inner products with kernels, and this is kernel *k*-means
- While *k*-means finds linear separators for the cluster boundaries, kernel *k*-means finds non-linear separators

Distances vs. Kernel Functions

- Mahalanobis distances:

$$d_A(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T A(\mathbf{x} - \mathbf{y})$$

- Inner products / kernels:

$$\kappa_A(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A\mathbf{y}$$

- Algorithms for constructing A learn both measures

From Linear to Nonlinear Learning

Consider the following *kernelized* problem

- You are given a kernel function $\kappa(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x})^T \varphi(\mathbf{y})$
- You want to run a metric learning algorithm in kernel space
    - Optimization algorithm cannot use the explicit feature vectors $\varphi(\mathbf{x})$
    - Must be able to compute the distance/kernel over arbitrary points (not just training points)
- Mahalanobis distance is of the form:

$$d_A(\mathbf{x}, \mathbf{y}) = (\varphi(\mathbf{x}) - \varphi(\mathbf{y}))^T A(\varphi(\mathbf{x}) - \varphi(\mathbf{y}))$$

- Kernel is of the form:

$$\kappa_A(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x})^T A \varphi(\mathbf{y})$$

- Can be thought of as a kind of *kernel learning* problem

Kernelization of ITML

- First example: ITML
- Recall the update for ITML

$$A_{t+1} = A_t + \beta_t A_t(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T A_t$$

  - Distance constraint over pair $(\mathbf{x}_i, \mathbf{x}_j)$
  - β_t computed in closed form
- How can we make this update *independent* of the dimensionality?

Kernelization of ITML

- Rewrite the algorithm in terms of inner products (kernel functions)

$$A_{t+1} = A_t + \beta_t A_t (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T A_t$$

- Inner products in this case: $\mathbf{x}_i^T A_t \mathbf{x}_j$

Kernelization of ITML

- Rewrite the algorithm in terms of inner products (kernel functions)

$$X^T A_{t+1} X = X^T A_t X + \beta_t X^T A_t X (\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^T X^T A_t X$$

- Entry (i, j) of $X^T A_t X$ is exactly $\mathbf{x}_i^T A \mathbf{x}_j = \kappa_A(\mathbf{x}_i, \mathbf{x}_j)$
- Denote $X^T A_t X$ as K_t, the *kernel matrix* at step t

$$K_{t+1} = K_t + \beta_t K_t (\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^T K_t$$

Kernel Learning

- Squared Euclidean distance in kernel space:

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \mathbf{x}_i^T \mathbf{x}_i + \mathbf{x}_j^T \mathbf{x}_j - 2\mathbf{x}_i^T \mathbf{x}_j$$

- Replace with kernel functions / kernel matrix:

$$\kappa(\mathbf{x}_i, \mathbf{x}_i) + \kappa(\mathbf{x}_j, \mathbf{x}_j) - 2\kappa(\mathbf{x}_i, \mathbf{x}_j) = K_{ii} + K_{jj} - 2K_{ij}$$

- Related to ITML, define the following optimization problem

$$\begin{array}{lll} \min_K & D_{\ell d}(K, K_0) & \\ \text{s.t.} & K_{ii} + K_{jj} - 2K_{ij} \leq u & \text{if } (i,j) \in \mathcal{S} \text{ [similarity constraints]} \\ & K_{ii} + K_{jj} - 2K_{ij} \geq \ell & \text{if } (i,j) \in \mathcal{D} \text{ [dissimilarity constraints]} \end{array}$$

  - $K_0 = X^T X$ is the input kernel matrix
  - To solve this, only the original kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ is required

Kernel Learning

- Bregman projections for the kernel learning problem:

$$K_{t+1} = K_t + \beta_t K_t(\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^T K_t$$

- Suggests a strong connection between the 2 problems
- **Theorem:** Let A^* be the optimal solution to ITML, and $A_0 = I$. Let K^* be the optimal solution to the kernel learning problem. Then $K^* = X^T A^* X$.
  - Solving the kernel learning problem is "equivalent" to solving ITML
  - So we can run entirely in kernel space
  - But, given two new points, how to compute distance?

[Davis, Kulis, Jain, Sra, and Dhillon; ICML 2007]

Induction with LogDet

- **Theorem:** Let A^* be the optimal solution to ITML, and let $A_0 = I$. Let K^* be the optimal solution to the kernel learning problem, and let $K_0 = X^T X$ be the input kernel matrix. Then

$$\begin{aligned} A^* &= I + XSX^T \\ S &= K_0^{-1}(K^* - K_0)K_0^{-1} \end{aligned}$$

  - Gives us a way to *implicitly* compute A^* once we solve for K^*
- Algorithm
  - Solve for K^*
  - Construct S using K_0 and K^*
  - Given two points $\mathbf{x}$ and $\mathbf{y}$, the kernel $\kappa_A(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A\mathbf{y}$ is computed as

$$\kappa_A(\mathbf{x}_i, \mathbf{x}_j) = \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i,j=1}^{n} S_{ij}\kappa(\mathbf{x}, \mathbf{x}_i)\kappa(\mathbf{x}_j, \mathbf{y})$$

[Davis, Kulis, Jain, Sra, and Dhillon; ICML 2007]

Kernelization of POLA

- Recall updates for POLA

$$\begin{aligned} \hat{A}_t &= A_t - y_t \alpha_t \mathbf{v}_t \mathbf{v}_t^T \\ A_{t+1} &= \hat{A}_t - \lambda_d \mathbf{u}_d \mathbf{u}_d^T \end{aligned}$$

  - $\mathbf{v}_t$ is the difference of 2 data points
  - $\mathbf{u}_d$ is the smallest eigenvector of $\hat{A}_t$
  - 1st update projects onto set C_t where hinge loss is zero (applied only when loss is non-zero)
  - 2nd update projects onto PSD cone C_a (applied only when $\hat{A}_t$ has negative eigenvalue)
- **Claim:** Analogous to ITML, $A^* = XSX^T$, where X is the matrix of data points
  - Prove this inductively

Kernelization of POLA

Projection onto C_t

- $A_t = XS_tX^T$
- Say the 2 data points are indexed by i and j
  - Then $\mathbf{v}_t = X(\mathbf{e}_i - \mathbf{e}_j)$
- Rewrite $A_t - y_t\alpha\mathbf{v}_t\mathbf{v}_t^T$ to get update from S_t to $\hat{S}_t$:

$$\begin{aligned}\hat{A}_t &= XS_tX^T - y_t\alpha_t X(\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^T X^T \\ &= X(S_t - y_t\alpha_t(\mathbf{e}_i - \mathbf{e}_j)(\mathbf{e}_i - \mathbf{e}_j)^T)X^T\end{aligned}$$

Projection onto C_a

- $\mathbf{u}_d$ is an eigenvector of $\hat{A}_t$, i.e.,

$$\begin{aligned}\hat{A}_t\mathbf{u}_d &= X\hat{S}_tX^T\mathbf{u} = \lambda_d\mathbf{u}_d \\ \mathbf{u}_d &= X\left(\frac{1}{\lambda_d}\hat{S}_tX^T\mathbf{u}_d\right) = X\mathbf{q}\end{aligned}$$

  - Construction for $\mathbf{q}$ non-trivial; involves kernelized Gram-Schmidt
  - Expensive (cubic in dimensionality)

General Kernelization Results

- Recent work by Chatpatanasiri et al. has shown additional kernelization results for
  - LMNN
  - Neighbourhood Component Analysis
  - Discriminant Neighborhood Embedding
- Other recent results show additional, general kernelization results
  - Xing et al.
  - Other regularizers (trace-norm)
- At this point, most/all existing Mahalanobis metric learning methods can be kernelized

Kernel PCA

- Setup for principal components analysis (PCA)
  - Let $X = [\mathbf{x}_1, ..., \mathbf{x}_n]$ be a set of data points
  - Typically assume data is centered, not critical here
  - Denote SVD of X as $X = U^T \Sigma V$
  - Left singular vectors in U corresponding to non-zero singular values are an orthonormal basis for the span of the $\mathbf{x}_i$ vectors
  - Covariance matrix is $C = XX^T = U^T \Sigma^T \Sigma U$, kernel matrix is $K = X^T X = V^T \Sigma^T \Sigma V$
- Standard PCA recipe
  - Compute SVD of X
  - Project data onto leading singular vectors U, e.g., $\tilde{\mathbf{x}} = U\mathbf{x}$

Kernel PCA

- Key result from the late 1990s: kernelization of PCA
  - Can also form projections using the kernel matrix
  - Allows one to avoid computing SVD
  - If $X = U^T \Sigma V$, then $U = \Sigma^{-1} V X^T$

$$U\mathbf{x} = \Sigma^{-1} V X^T \mathbf{x}$$

  - Computation involves inner products $X^T\mathbf{x}$, eigenvectors V of the kernel matrix, and eigenvalues of the kernel matrix
- Relation to Mahalanobis distance methods
  - Kernel PCA allows one to implicitly compute an orthogonal basis U of the data points, and to project arbitrary data points onto this basis
  - For a data set of n points, dimension of basis is at most n
  - Projecting onto U results in an n-dimensional vector

Using kernel PCA for metric learning

- Given a set of points in kernel space $X = [\varphi(\mathbf{x}_1), ..., \varphi(\mathbf{x}_n)]$
  - Form a basis U and project data onto that basis to form $\tilde{X} = [\tilde{x}_1, ..., \tilde{x}_n] = [U\varphi(\mathbf{x}_1), ..., U\varphi(\mathbf{x}_n)]$ using kernel PCA
- Consider a general unconstrained optimization problem f that is a function of kernel function values, i.e.

$$f(\{\varphi(\mathbf{x}_i)^T A\varphi(\mathbf{x}_j)\}_{i,j=1}^n)$$

- Associated minimization

$$\min_{A \succeq 0} f(\{\varphi(\mathbf{x}_i)^T A\varphi(\mathbf{x}_j)\}_{i,j=1}^n)$$

- **Theorem:** The optimal value of the above optimization is the same as that of

$$\min_{A' \succeq 0} f(\{\tilde{x}_i^T A' \tilde{x}_j\}_{i,j=1}^n)$$

where A' is $n \times n$.

[Chatpatanasiri, Korsrilabutr, Tangchanachaianan, and Kijsirikul; ArXiV 2008]

Consequences

- Any Mahalanobis distance learning method that is unconstrained and can be expressed as a function of learned inner products can be kernelized
- Examples
  - Neighbourhood Components Analysis
  - LMNN (write as unconstrained via the hinge loss)
  - Discriminant neighborhood embedding
- Generalizing to new points
  - For a new point $\varphi(\mathbf{x})$, construct $\tilde{\mathbf{x}}$ and use Mahalanobis distance with learned matrix A'
- Algorithms
  - Exactly the same algorithms employed as in linear case

Extensions

- Chatpatanasiri et al. considered extensions for low-rank transformations
  - Also showed benefits of kernelization in several scenarios
- Recent results (Jain et al.) have shown complementary results for *constrained* optimization problems
  - ITML is a special case of this analysis
  - Other methods follow easily, e.g., methods based on trace-norm regularization
- Now most Mahalanobis metric learning methods have been shown to be kernelizable

Scalability in Kernel Space

- In many situations, dimensionality d and the number of data points n is high
  - Typically, linear Mahalanobis metric learning methods scale as $O(d^2)$ or $O(d^3)$
  - Kernelized Mahalanobis methods scale as $O(n^2)$ or $O(n^3)$
  - What to do when both are large?
- Main idea: restrict the basis used for learning the metric
  - Can be applied to most methods

Scalability with the kernel PCA approach

- Recall the kernel PCA approach
  - Project onto U, the top n left singular vectors
  - Instead, project onto the top r left singular vectors
  - Proceed as before
- Similar approach can be used for ITML
  - The learned kernel is of the form

$$\kappa_A(\mathbf{x}_i, \mathbf{x}_j) = \kappa(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i,j=1}^{n} S_{ij}\kappa(\mathbf{x}, \mathbf{x}_i)\kappa(\mathbf{x}_j, \mathbf{y})$$

  - Restrict S to be $r \times r$ instead of $n \times n$, where $r < n$ data points are chosen
  - Rewrite optimization problem using this form of the kernel
  - Constraints on learned distances are still linear, so method can be generalized
- Both approaches can be applied to very large data sets
  - Example: ITML has been applied to data sets of nearly 1 million points (of dimensionality 24,000)

Nearest neighbors with Mahalanobis metrics

- Once metrics are learned, k-nn is typically used
  - k-nn is expensive to compute
  - Must compute distances to all n training points
- Recent methods attempt to speed up NN computation
  - Locality-sensitive hashing
  - Ball trees
- One challenge: can such methods be employed even when algorithms are used in kernel space?
  - Recent work applied in computer vision community has addressed this problem for fast image search

Other non-linear methods

- Recall that kernelized Mahalanobis methods try to learn the distance function

$$\|G\varphi(\mathbf{x}) - G\varphi(\mathbf{y})\|_2^2$$

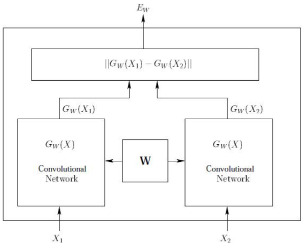
- Chopra et al. learn the non-linear distance

$$\|G_W(\mathbf{x}) - G_W(\mathbf{y})\|_2^2$$

  - G_W is a non-linear function
  - Application was face verification
  - Algorithmic technique: convolutional networks

[Chopra, Hadsell, and LeCun; CVPR 2005]

Other non-linear methods

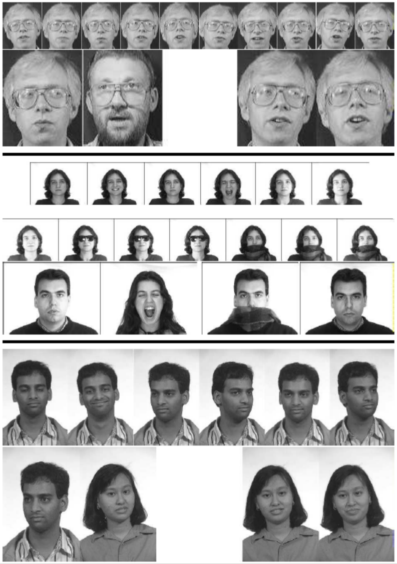


- Setup uses relative distance constraints
  - Denote D_{ij} as the mapped distance between points i and j
  - Let $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ be a tuple such that $D_{ij} < D_{ik}$ desired
  - The authors define a loss function for each triple of the form

$$Loss = \alpha_1 D_{ij} + \alpha_2 \exp(-\alpha_3 \sqrt{D_{ik}})$$

  - Minimize the sum of the losses over all triples
- Metric is trained using a convolutional network with a Siamese architecture from the pixel level

Other non-linear methods



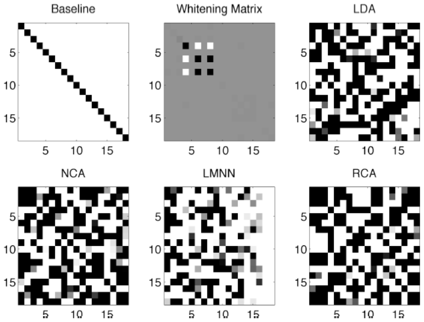
Application: Learning Music Similarity

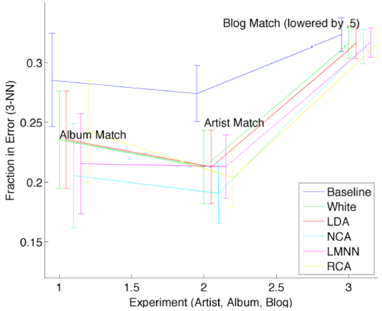
Comparison of metric learning methods for learning music similarity

- MP3s downloaded from a set of music blogs
  - After pruning: 319 blogs, 164 artists, 74 distinct albums
  - Thousands of songs
- The Echo Nest used to extract features for each song
  - Songs broken up into segments (80ms to a few seconds)
  - Mean segment duration
  - Track tempo estimate
  - Regularity of the beat
  - Estimation of the time signature
  - Overall loudness estimate of the track
  - Estimated overall tatum duration
  - In total, 18 features extracted for each song
- Training done via labels based on blog, artist, and album (separately)

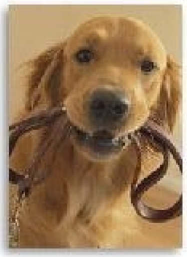
[Slaney, Weinberger, and White; ISMIR 2008]

Application: Learning Music Similarity





Application: Object Recognition

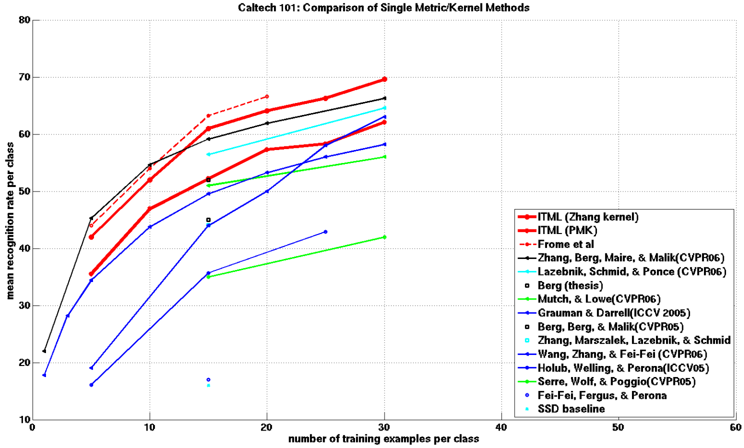


- Several metric learning methods have been evaluated on the Caltech 101 dataset, a benchmark for object recognition set size

Application: Object Recognition

- Used the Caltech-101 data set
  - Standard benchmark for object recognition
  - Many many results for this data set
  - 101 classes, approximately 4000 total images
- Learned metrics over 2 different image embeddings for ITML: pyramid match kernel (PMK) embedding and the embedding from Zhang et al, 2006
- Also learned metrics via Frome et al's local distance function approach
- Computed *k*-nearest neighbor accuracy over varying training set size and compared to existing results

Application: Object Recognition



Results: Clarify

- Representation: System collects program features during run-time
  - Function counts
  - Call-site counts
  - Counts of program paths
  - Program execution represented as a vector of counts
- Class labels: Program execution errors
- Nearest neighbor software support
  - Match program executions
  - Underlying distance measure should reflect this similarity
- Results
  - LaTeX Benchmark: Error drops from 30% to 15%
  - LogDet is the best performing algorithm across all benchmarks

[Davis, Kulis, Jain, Sra, and Dhillon; ICML 2007]

Application: Human Body Pose Estimation

Pose Estimation

- 500,000 synthetically generated images
- Mean error is 34.5 cm per joint between two random images

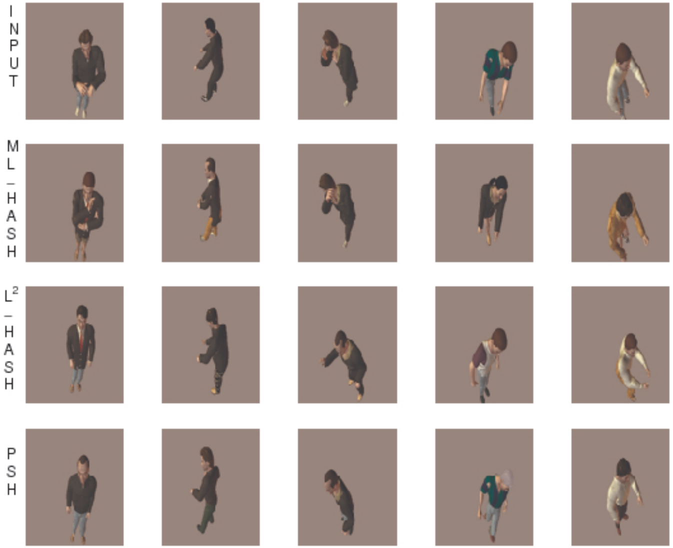
Pose Estimation Results

Method	m	k=1
L_2 linear scan	24K	8.9
L_2 hashing	24K	9.4
PSH, linear scan	1.5K	9.4
PCA, linear scan	60	13.5
PCA+LogDet, lin. scan	60	13.1
LogDet linear scan	24K	8.4
LogDet hashing	24K	8.8

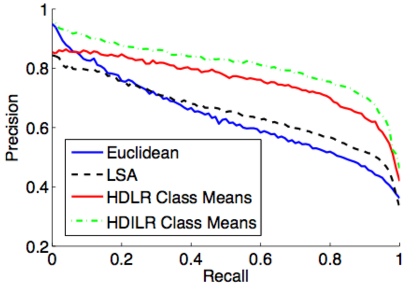
- Error above given is mean error in cm per joint
- Linear scan requires 433.25 seconds per query; hashing requires 1.39 seconds per query (hashing searches 0.5% of database)

[Jain, Kulis, and Grauman; CVPR 2008]

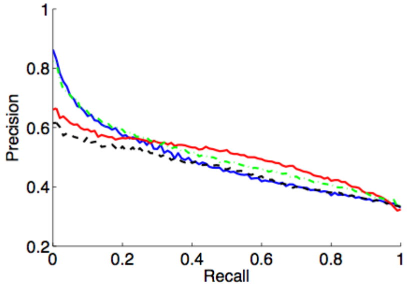
Pose Estimation Results



Application: Text Retrieval



(a) Classic3



(a) Newsgroups Politics Subset

[Davis and Dhillon; SIGKDD 2008]

Summary and Conclusions

- Metric learning is a mature technology
  - Complaints about scalability in terms of dimensionality or number of data points no longer valid
  - Many different formulations have been studied, especially for Mahalanobis metric learning
  - Online vs offline settings possible
- Metric learning has been applied to many interesting problems
  - Language problems
  - Music similarity
  - Pose estimation
  - Image similarity and search
  - Face verification

Summary and Conclusions

- Metric learning has interesting theoretical components
  - Analysis of online settings
  - Analysis of high-dimensional (kernelized) settings
- Metric learning is still an interesting area of study
  - Learning multiple metrics over data sets
  - New applications
  - Formulations that integrate better with problems other than k-nn
  - Improved algorithms for better scalability
  - ...