Accuracy of Distance Metric Learning Algorithms

Frank Nielsen
École Polytechnique
Route de Saclay
91128, Palaiseau cedex, France
Sony CSL
Tokyo, Japan
nielsen@lix.polytechnique.fr

Aurélien Sérandour
École Polytechnique
Route de Saclay
91128, Palaiseau cedex, France
aurelien.serandour@polytechnique.org

ABSTRACT

In this paper, we wanted to compare distance metric-learning algorithms on UCI datasets. We wanted to assess the accuracy of these algorithms in many situations, perhaps some that they were not initially designed for. We looked for many algorithms and chose four of them based on our criteria. We also selected six UCI datasets. From the data's labels, we create similarity dataset that will be used to train and test the algorithms. The nature of each dataset is different (size, dimension), and the algorithms' results may vary because of these parameters. We also wanted to have some robust algorithms on dataset whose similarity is not perfect, whose the labels are no well defined. This occurs in multi-labeled datasets or even worse in human-built ones. To simulate this, we injected contradictory data and observed the behavior of the algorithms. This study seeks for a reliable algorithm in such scenarios keeping in mind future uses in recommendation processes.

Categories and Subject Descriptors

I.5.3 [**Pattern Recognition**]: Clustering Algorithms, Similarity measures; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra, Singular value decomposition; G.1.6 [**Optimization**]: Constrained optimization

Keywords

Metric learning, Mahalanobis distance, Bregman divergences

1. INTRODUCTION

In many unsupervised learning problems, algorithms tend to find cluster to separate data, to label them. However there is sometimes no perfect boundaries between these assumed groups. They can easily overlap. For example, in music, even if some labels exist (classical, jazz, rock etc.), one can see each song as a combination of them. This can be annoying in a recommendation process because it becomes impossible to rely on these labels. In order to remove the definition of labels and avoid song clustering during the recommendation, one can see the problem as a similarity one. Given an entry point, a user can jump from a song to another in a logical way, respecting some similarity constrains. If a user is listening to a song, the most similar song can be recommended as the next one in an automatically generated playlist. Of course the area is not bound to music and can be applied to any recommendation problem.

The similarity is a binary evaluation: similar or dissimilar. Representing the similarity between two data as a distance function and a threshold is the most convenient way. Above a threshold, pair is a dissimilar one, under it, it is a similar one. So learning this distance function should give a solution to the problem. Many algorithms have been proposed, focusing on Mahanalobis distance. We decided to compare them on several datasets.

The similarity between data is not a mathematically defined attribute in music for example. It is more about feeling. So the sets have to be human-built ones. Unfortunately the human factor ensures that some randomness and inconsistencies will occur. This is an essential parameter in the recommendation process and it shouldn't be neglected.

2. NOTATION USED

We will use these notations all along:

$$
\begin{aligned}
d &= \text{dimension of the space}\\
\mathcal{S} &= \{(x_i, x_j) | x_i \text{ and } x_j \text{ are similar}\}\\
\mathcal{D} &= \{(x_i, x_j) | x_i \text{ and } x_j \text{ are dissimilar}\}\\
N &= \text{total number of points}\\
\mathcal{M}_d(\mathbb{R}) &= d \times d \text{ matrices in } \mathbb{R}\\
\mathcal{S}_d(\mathbb{R}) &= \left\{ M \in \mathcal{M}_d(\mathbb{R}) | M = M^T \right\}\\
\mathcal{S}_d^+(\mathbb{R}) &= \left\{ M \in \mathcal{S}_d(\mathbb{R}) | \forall X \in \mathbb{R}^d,\, X^T M X \geqslant 0 \right\}\\
\mathcal{S}_d^{++}(\mathbb{R}) &= \left\{ M \in \mathcal{S}_d(\mathbb{R}) | \forall X \in \mathbb{R}^d,\, X^T M X > 0 \right\}\\
\|\cdot\|_F &= \text{ Frobenius norm on matrices}\\
\|\cdot\|_A &= \text{ Mahalanobis distance with matrix A}\\
A \succeq 0 &\text{ means } A \in \mathcal{S}_d^+(\mathbb{R})
\end{aligned}
$$

3. MODUS OPERANDI

3.1 Datasets

DMMT'09, June 28, 2009, Paris.

We chose six datasets from the UCI[1] database.

- Iris[2]: low dimensional dataset
- Ionosphere[3]: high dimensional large dataset.
- Wine[4]: middle class dataset.
- Wisconsin Breast Cancer (WDBC)[5]: high dimensional large dataset.
- Soybean small[6]: high dimensional dataset with few data and many classes.
- Balance-scale[7]: low dimensional large dataset.

The details of these datasets can be seen in Tab. 1. Note that the final size of the similarity dataset is $\frac{N(N-1)}{2}$ where N is the number of points in the first one.

Table 1: Datasets attributes

DATASET	DIMENSION	SIZE	NUMBER OF CLASSES
Iris	4	150	3
Ionosphere	34	351	2
Wine	13	178	3
WDBC	32	569	2
Soybean small	35	47	4
Balance-scale	4	625	3

To remain as close as possible to music recommendation, we chose to use unlabeled datasets (even if this means removing labels on our own). In this study, only similarity is given.

3.2 Algorithms

We chose to compare the distance matrices generated from four algorithms based on our constraints: unlabeled data and similarity sets. The algorithms are:

- no algorithm: the identity matrix or Euclidean distance
- Xing's algorithm [7]: an iterative algorithm
- Information-Theoretic Metric Learning [3]: an iterative algorithm
- Coding similarity [6]

We chose them because of their different ways to formulate the similarity problem. This would give us an overview of what can be done today. We also studied other algorithms but decided not to incude them in this paper (see Appendix).

[1]http://archive.ics.uci.edu/ml/
[2]http://archive.ics.uci.edu/ml/datasets/Iris
[3]http://archive.ics.uci.edu/ml/datasets/Ionosphere
[4]http://archive.ics.uci.edu/ml/datasets/Wine
[5]http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)
[6]http://archive.ics.uci.edu/ml/datasets/Soybean+(Small)
[7]http://archive.ics.uci.edu/ml/datasets/Balance+Scale

3.2.1 Xing's algorithm

This algorithm is the simplest someone can think about to solve the problem. The general idea is to minimize the distance between similar points and dissimilar ones. For that, we consider a distance matrix $A \in \mathcal{S}_d^+(\mathbb{R})$ and the following optimization problem.

$$\begin{aligned} \min_A & \sum_{(x_i,x_j)\in\mathcal{D}} \|x_i - x_j\|_A \\ \text{subject to } & \sum_{(x_i,x_j)\in\mathcal{S}} \|x_i - x_j\|_A^2 \leqslant 1 \\ & A \succeq 0 \end{aligned}$$

This formulation allows to put any condition we want on A. For example we can enforce A to be diagonal. This way, we can prevent overfitting, but perhaps decrease accuracy at the same time.

The optimization problem used to learn a full matrix is slighty different, but it is the same idea: move closer similar points and separate dissimilar ones.

$$\begin{aligned} \max_A \ g(A) = & \sum_{(x_i,x_j)\in\mathcal{S}} \|x_i - x_j\|_A^2 \\ \text{subject to } f(A) = & \sum_{(x_i,x_j)\in\mathcal{D}} \|x_i - x_j\|_A \geqslant 1 \\ & A \succeq 0 \end{aligned}$$

The details of the algorithm are described in the Appendix.

The main drawback is that the convergence is not sure. Sometimes, depending on the dataset or the initial conditions, it may not be able to gives a good result and get stuck in a loop where each iteration step gives a new matrix far from the previous one. On some datasets, it may find $A = 0$, which is not wanted.

In this paper, the algorithm was only run to learn full matrices.

3.2.2 Information Theoretic Metric Learning

This paper is one of the last published on this subject. It contains several new methods. The problem has evolved since its first publication. The last one, which is the one we used, is:

$$\begin{aligned} & \min_{A\succeq 0} D_{ld}(A, A_0) + \gamma \cdot D_{ld}(\text{diag}(\xi), \text{diag}(\xi_0)) \\ \text{subject to } & \|x_i - x_j\|_A \leqslant \xi_{i,j} \text{ if } (x_i, x_j) \in \mathcal{S} \\ & \|x_i - x_j\|_A \geqslant \xi_{i,j} \text{ if } (x_i, x_j) \in \mathcal{D} \end{aligned}$$

The Kullback-Leibler distance (or KL divergence) is a statistical distance between two distributions. The formula is:

$$D_{KL}(P\|Q) = \text{KL}(p_P(x)|p_Q(x)) = \int_\Omega P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx$$
$$\text{and KL}(p(x;A_0)\|p(x;A)) = \frac{1}{2} D_{ld}(A, A_0)$$

ξ_0 is the set of thresholds defining the bound between similarity and dissimilarity and $\xi_{i,j}$ is the threshold for the pair (i, j), which can be in $\mathcal{S}$ or $\mathcal{D}$. γ controls the tradeoff between learning a matrix close to an arbitrary matrix A_0 and modifying the pre-computed threshold. This problem

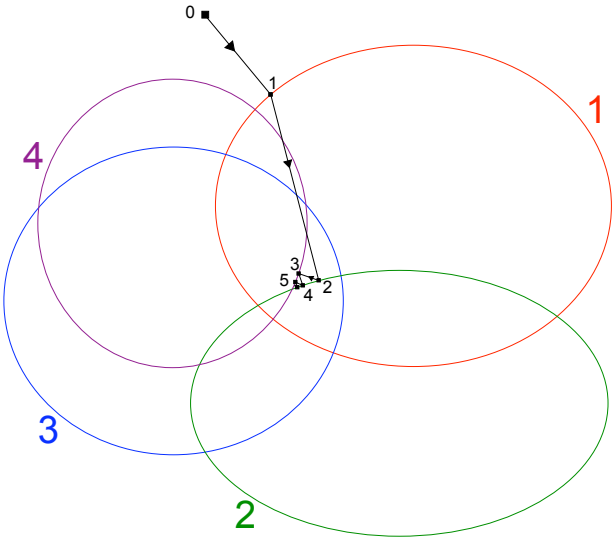

Figure 1: Projections on convex subspaces and convergence to a common intersection point (in the limit case)

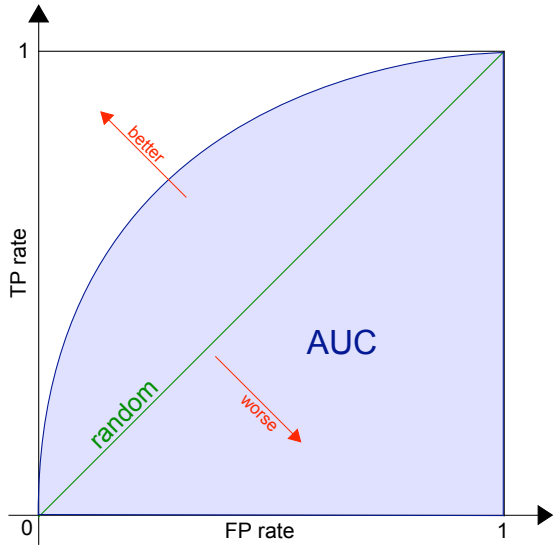

Figure 2: ROC curve

has many parameters which can be used to compute better results. However, it needs to decide at the beginning of the algorithm the values of A_0, ξ and γ. It is very difficult to guess them *a priori*.

The algorithm is an iterative method on each pair of $\mathcal{S}$ and $\mathcal{D}$. It performs successive projections on subspaces $\mathcal{S}_d^{(i,j)} = \left\{M \in \mathcal{S}_d^+ \,|\, \|x_i - x_j\|_M^2 \leqslant u\right\}$ (Fig. 1)[8] . The convergence is proven thanks to Bregman's research[2] in this field[9].

There is also an online version of this algorithm. We didn't report the result here since the accuracy was worse than the offline one. We can also enforce some constraints on A but for the same reasons, we kept with full matrices.

There are some problems such as the one in Xing's algorithm. However, they occur less frequently, almost never in fact.

3.2.3 *Coding Similarity*

This algorithm originally does not intend to learn a distance matrix. Finding a distance Matrix is just a consequence of it. The goal is to find a function that will estimate the similarity between two data. Coding similarity is defined by the amount of information a data contains about another: the similarity $\text{codsim}(x, x')$ is the information that x conveys about x'. This is a distribution based method. This amount of information is evaluated by the difference of coding length between two distribution: one where the *real* relation between x and x' is respected, and one there it is not. Note that this algorithm only use similar pairs. Coding length is $\text{cl}(x)$ is $-\log(p(x))$. The final definition is:

$$\begin{aligned}\text{codsim}(x, x') &= \text{cl}(x) - \text{cl}(x|x', H_1) \\ &= \log(p(x|x', H_1)) - \log(p(x))\end{aligned}$$

[8]for an animated applet, see http://www.lix.polytechnique.fr/~nielsen/BregmanProjection/

[9]for details, please see Stephen Boyd and Jon Dattorro course at Stanford University http://www.stanford.edu/class/ee392o/alt_proj.pdf

This algorithm can also performs dimension reduction to avoid overfitting.

Although it doesn't require iteration, the computation can be expensive (matrix inversion). Futhermore, since it requires symmetric matrix inversion, there can be some issues in this step. To avoid this, zero eigen values were set to a small amount.

In our tests, no dimension reduction was computed.

3.3 Tests and evaluation method

From each dataset, given the class of each data, we label each possible pair by similar or dissimilar pairs. These are the input of the algorithms. We performed a two fold cross-validation based on it. We also need a threshold to evaluate the similarity. However, the algorithms we use do not give one. So in the evaluation process, we want to remove the choice of one threshold. Given the distance matrix and several thresholds, we compute several confusion matrices. The threshold are chosen so that we describe the entire spectrum of interesting values: from the one that maps everything to dissimilar to the one that maps everything to similar. Then the Receiver Operating Characteristic (ROC) curve is computed and the accuracy of the model is given by the area under the ROC curve (AUC) (Figure 2).

For ITML, we need to initialize ξ_0. We set them as suggested the authors although it is a totally arbitrary choice: similarity at 5% of all pairs' distance distribution, dissimilarity at 95%.

We also wanted to study the effect of incoherent data on the overall result. The sets give perfect similarity whereas human-built ones may not pretend to this property. So we chose to insert some contradictory data by "flipping" the similarity of some pairs. The method does not add new pairs but modify the existing ones, so that the dataset does not have contradictory pairs but contains similarity evaluation errors.

Since we generated the inconsistencies at random, we chose to exclude them from the test set. In a real dataset, human-

built one, we cannot have this choice. However, this only reduce the results and does not help to compare the algorithms.

3.4 Software description

In addition to the AUC computation for both train and test sets, we created an application able to performs many tests (see Figure 3). It displays the ROC curve, the Precision and Recall curve and confusion matrices. The second curve gives an interesting threshold to separate the data. It can performs:

- Analysis of variance
- student's t-test: it compares the result of two algorithms and determines if there is a significant difference between them
- Tukey's test: same as student's t-test but compares several algorithms at the same time
- Spearman's rank correlation: it estimates if the data is correctly sorted with the computed distance.
- p-value

4. ALGORITHMS' DECRIPTION

4.1 Xing's algorithm

4.1.1 Algorithm for full matrix

Here we present the algorithm we derived to compute the full matrix.

Algorithm 1 Xing's algorithm for full matrix

repeat
  repeat
    $A := \text{argmin}_{A'} \left\{ \|A' - A\|_F : A' \in \mathcal{C}_1 \right\}$
    $A := \text{argmin}_{A'} \left\{ \|A' - A\|_F : A' \in \mathcal{C}_2 \right\}$
  until A converges
  $A := A + \alpha(\nabla_A g(A))_{\perp \nabla_A f}$
until convergence

where:

$$\mathcal{C}_1 = \left\{ A | \sum_{(x_i,x_j)\in\mathcal{S}} \|x_i - x_j\|_A^2 \leqslant 1 \right\}$$
$$\mathcal{C}_2 = \mathcal{S}_d^+(\mathbb{R})$$

Projection on $\mathcal{C}_1$.

$$\begin{aligned}
\delta &= \sum_{(x_i,x_j)\in\mathcal{S}} \|x_i - x_j\|_{A'}^2 \\
&= \sum_{(x_i,x_j)\in\mathcal{S}} \sum_{k=0}^{d} (x_{ik} - x_{jk}) \left[\sum_{p=0}^{d} a'_{kp}(x_{ip} - x_{jp}) \right] \\
&= \sum_{k,p} a'_{kp} \left[\sum_{(x_i,x_j)\in\mathcal{S}} (x_{ik} - x_{jk})(x_{ip} - x_{jp}) \right] \\
&= \sum_{k,p} a'_{kp} \beta_{kp}^{\mathcal{S}} \\
\text{where } \beta_{kp}^{\mathcal{S}} &= \sum_{(x_i,x_j)\in\mathcal{S}} (x_{ik} - x_{jk})(x_{ip} - x_{jp})
\end{aligned}$$

Now set up the Lagrangian:

$$\begin{aligned}
\mathcal{L}(\lambda) &= \left\|A - A'\right\|_F^2 + \lambda \left[\sum_{(x_i,x_j)\in\mathcal{S}} \|x_i - x_j\|_{A'}^2 - 1 \right] \\
&= \sum_{k,p} \left(a'_{kp} - a_{kp}\right)^2 + \lambda \left[\sum_{k,p} a'_{kp} \beta_{kp}^{\mathcal{S}} - 1 \right]
\end{aligned}$$

calculate the derivatives and solve the system $\mathfrak{S}$:

$$\begin{aligned}
\mathfrak{S} &\Leftrightarrow \begin{cases} \frac{\partial \mathcal{L}}{\partial a'_{kp}} &= 2\left(a'_{kp} - a_{kp}\right) + \lambda \beta_{kp}^{\mathcal{S}} = 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} &= \left[\sum_{k,p} a'_{kp} \beta_{kp}^{\mathcal{S}} \right] - 1 = 0 \text{ or } \lambda = 0 \end{cases} \\
&\Leftrightarrow \begin{cases} a'_{kp} = a_{kp} - \frac{\lambda \beta_{kp}^{\mathcal{S}}}{2} \\ \sum_{k,p} a'_{kp} \beta_{kp}^{\mathcal{S}} = 1 \text{ or } \lambda = 0 \end{cases} \\
&\Leftrightarrow \begin{cases} a'_{kp} = a_{kp} - \frac{\lambda \beta_{kp}^{\mathcal{S}}}{2} \\ \sum_{k,p} \left[a_{kp} - \frac{\lambda \beta_{kp}^{\mathcal{S}}}{2} \right] \beta_{kp}^{\mathcal{S}} = 1 \text{ or } \lambda = 0 \end{cases} \\
&\Leftrightarrow \begin{cases} a'_{kp} = a_{kp} - \frac{\lambda \beta_{kp}^{\mathcal{S}}}{2} \\ \sum_{k,p} -\frac{\lambda \beta_{kp}^{\mathcal{S}2}}{2} = 1 - \sum_{k,p} a_{kp} \beta_{kp}^{\mathcal{S}} \text{ or } \lambda = 0 \end{cases} \\
&\Leftrightarrow \begin{cases} a'_{kp} &= a_{kp} - \frac{\lambda \beta_{kp}^{\mathcal{S}}}{2} \\ \lambda &= 2 \frac{\left(\sum_{k,p} a_{kp} \beta_{kp}^{\mathcal{S}} \right) - 1}{\sum_{k,p} \beta_{kp}^{\mathcal{S}2}} \text{ or } \lambda = 0 \end{cases} \\
&\Leftrightarrow \begin{cases} a'_{kp} &= a_{kp} - \beta_{kp}^{\mathcal{S}} \frac{\left(\sum_{k,p} a_{kp} \beta_{kp}^{\mathcal{S}} \right) - 1}{\sum_{k,p} \beta_{kp}^{\mathcal{S}2}} \\ \lambda &= 2 \frac{\left(\sum_{k,p} a_{kp} \beta_{kp}^{\mathcal{S}} \right) - 1}{\sum_{k,p} \beta_{kp}^{\mathcal{S}2}} \text{ or } \lambda = 0 \end{cases}
\end{aligned}$$

The update is:

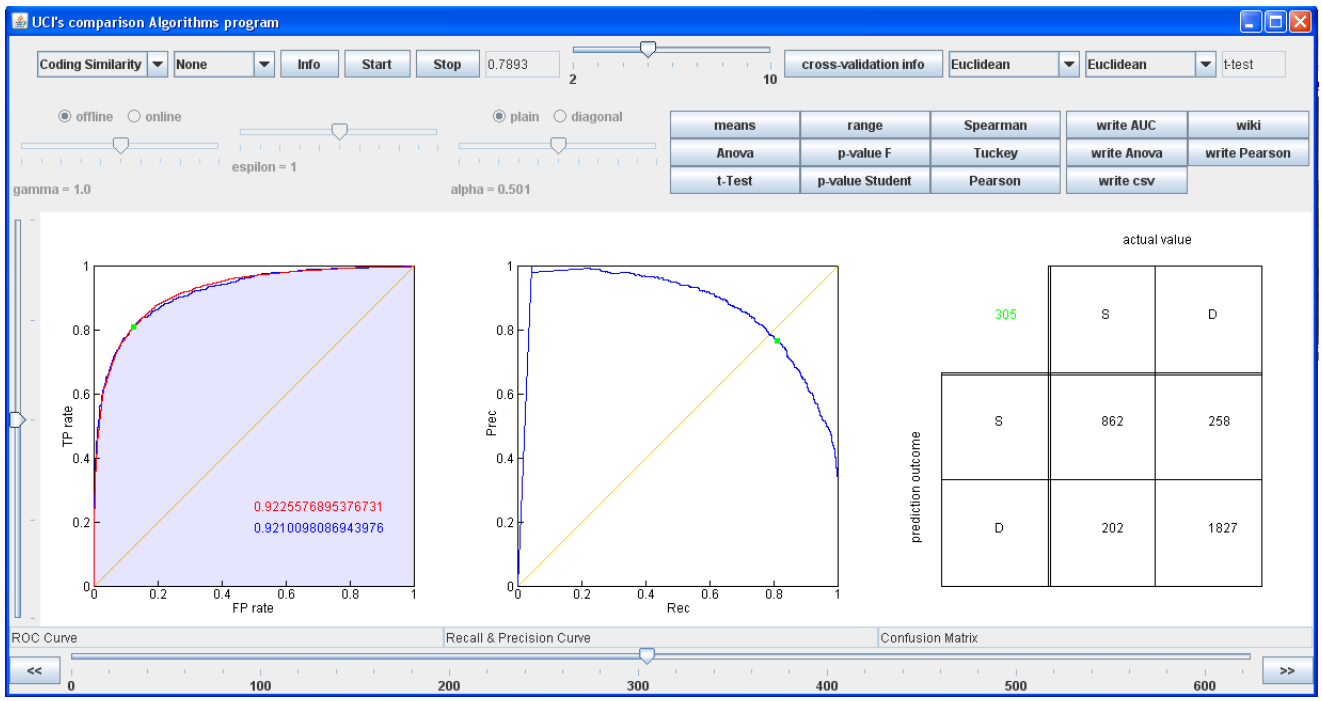

Figure 3: Snapshot of the software for benchmarking metric learning methods

$$a'_{kp} = a_{kp} + \beta^{\mathcal{S}}_{kp} \frac{1 - \sum_{k,p} a_{kp}\beta^{\mathcal{S}}_{kp}}{\sum_{k,p} \beta^{\mathcal{S}2}_{kp}} \text{ if } f(A) > 1$$

$$\text{where } \beta^{\mathcal{S}}_{kp} = \sum_{(x_i,x_j)\in\mathcal{S}} (x_{ik} - x_{jk})(x_{ip} - x_{jp}) = \beta^{\mathcal{S}}_{pk}$$

We can find an interesting formulation since:

$$\text{if } \beta^{\mathcal{S}} = \left(\beta^{\mathcal{S}}_{kp}\right)_{k,p} = \sum_{(x_i,x_j)\in\mathcal{S}} (x_i - x_j)(x_i - x_j)^T$$

$$\sum_{k,p} \beta^{\mathcal{S}2}_{kp} = \left\|\beta^{\mathcal{S}}\right\|_F^2$$

$$\sum_{k,p} a_{kp}\beta^{\mathcal{S}}_{kp} = \sum_{k,p} a_{kp}\beta^{\mathcal{S}}_{pk} = \text{trace}\left(A\beta^{\mathcal{S}}\right)$$

We get the final formulation:

$$\boxed{A' = A + \beta^{\mathcal{S}} \frac{1 - \text{trace}\left(A\beta^{\mathcal{S}}\right)}{\|\beta^{\mathcal{S}}\|_F^2}} \text{ if } f(A) > 1 \text{ , else } \boxed{A' = A}$$

Projection on $\mathcal{C}_2$.

set the negative eigen values to 0:

$$\boxed{\begin{array}{l} A = X\Lambda X^T \text{ where } \Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d) \\ \Lambda' = \text{diag}\left(\max(\lambda_1, 0), \max(\lambda_2, 0), \ldots, \max(\lambda_d, 0)\right) \\ \text{set } A = A' = X\Lambda' X^T \end{array}}$$

Here it can be difficult to avoid $A = 0$. This was a common issue in this algorithm.

Gradient ascent.

$$\overrightarrow{\nabla}_A g(A) = \begin{pmatrix} \frac{\partial g}{\partial a_{1,1}} & \cdots & \frac{\partial g}{\partial a_{1,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial g}{\partial a_{n,1}} & \cdots & \frac{\partial g}{\partial a_{n,n}} \end{pmatrix} (A)$$

$$\text{where } \frac{\partial g}{\partial a_{kp}} = \frac{\partial}{\partial a_{kp}} \left[\sum_{\mathcal{D}} \sqrt{\sum_{u,v} a_{uv}(x_{iu} - x_{ju})(x_{iv} - x_{jv})}\right]$$

$$= \sum_{\mathcal{D}} \frac{(x_{iu} - x_{ju})(x_{iv} - x_{jv})}{2\sqrt{\sum_{u,v} a_{u,v}(x_{iu} - x_{ju})(x_{iv} - x_{jv})}}$$

$$\text{so } \overrightarrow{\nabla}_A g(A) = \sum_{(x_i,x_j)\in\mathcal{D}} \frac{(x_i - x_j)(x_i - x_j)^T}{2\|x_i - x_j\|_A}$$

$$\text{where } \frac{\partial f}{\partial a_{kp}} = \frac{\partial}{\partial a_{kp}} \left[\sum_{\mathcal{S}} \sum_{u,v} a_{uv}(x_{iu} - x_{ju})(x_{iv} - x_{jv})\right]$$

$$= \sum_{\mathcal{S}} (x_{iu} - x_{ju})(x_{iv} - x_{jv}) = \beta^{\mathcal{S}}_{uv}$$

$$\text{so } \overrightarrow{\nabla}_A f(A) = \beta^{\mathcal{S}}$$

We have the final formulation:

$$\left[\overrightarrow{\nabla}_A g(A)\right]_{\perp \overrightarrow{\nabla}_A f} = \overrightarrow{\nabla}_A g(A) - \left[\frac{\overrightarrow{\nabla}_A g \cdot \overrightarrow{\nabla}_A f}{\left\|\overrightarrow{\nabla}_A f\right\|^2} \overrightarrow{\nabla}_A f\right] (A)$$

which can be written:

$$\left[\overrightarrow{\nabla}_A g(A)\right]_{\perp \overrightarrow{\nabla}_A f} = \overrightarrow{\nabla}_A g(A) - \frac{\overrightarrow{\nabla}_A g(A) \cdot \beta^{\mathcal{S}}}{\|\beta^{\mathcal{S}}\|_F^2} \beta^{\mathcal{S}}$$

4.2 Algorithm for diagonal matrix

If we want to learn a diagonal matrix $A = \text{diag}\,(a_{1,1} \dots a_{n,n})$, we just look for minimizing the function $h(A)$:

$$h(A) = h\,(a_{1,1} \dots a_{n,n})$$

$$= \sum_{(x_i,x_j)\in\mathcal{S}} \|x_i - x_j\|_A^2 - \log\left(\sum_{(x_i,x_j)\in\mathcal{D}} \|x_i - x_j\|_A\right)$$

We use a Raphson-Newton method to find the minimum of h. However, because of the log function, we cannot have $\exists i, a_{i,i} < 0$ or $\forall i, a_{i,i} = 0$. So monitoring the Newton-Raphson algorithm prevents this case (Fig. 4).

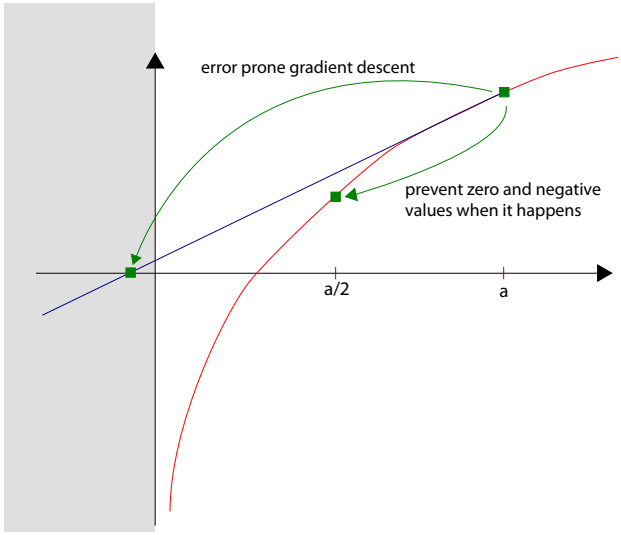

Figure 4: monitoring the Newton-Raphson algorithm

5. CODING SIMILARITY

The main advantage of this method is that there is no iteration. It is very fast in low dimension. However, due to the matrices inversions, high dimension problems can be very slow. Also notice that due to precision errors, in the program I had to force the distance matrix to be perfectly symmetric thanks to the update: $A = \frac{A+A^T}{2}$.

We also modified this algorithm in order to learn a diagonal matrix. We finally removed that option, the results were not significantly different.

6. RAW RESULTS

The tables below gives the AUC for each algorithm and for several percentages of errors in the dataset. We only put these results here since the ones from the other tests weren't as relevant as these ones. The noise rate corresponds to the percentage of input pairs whose similarity label was "flipped".

Algorithm 2 Coding Similarity algorithm

1: let $L = \#\mathcal{S}$
2: $Z = \frac{1}{2L}\sum_{\mathcal{S}}(x_i - x_j)$
3: remove Z to each data
4: $\Sigma_x = \frac{1}{2L}\sum_{\mathcal{S}}\left[x_i x_i^T + x_j x_j^T\right]$
5: $\Sigma_{xx'} = \frac{1}{2L}\sum_{\mathcal{S}}\left[x_i x_j^T + x_j x_i^T\right]$
6: $\Sigma_\Delta = \frac{\Sigma_x - \Sigma_{xx'}}{2}$
7: Distance matrix A is $(4\Sigma_\Delta)^{-1} = [2\,(\Sigma_x - \Sigma_{xx'})]^{-1}$

6.1 Ionosphere

Table 2: Ionosphere results

noise rate	Euclidean	Xing	ITML	CodeSim
0	0.645	**0.8499**	0.7913	0.7787
5	0.645	**0.8322**	0.5322	0.7752
10	0.645	0.7208	0.5106	**0.7728**
20	0.645	0.6273	0.4988	**0.7683**
25	0.645	0.7175	-	**0.7662**
30	0.645	0. 6792	-	**0.7639**

The results are written in table 2. The Euclidean norm gives poor results, so learning a distance is interesting. Xing algorithm performs well but shows low robustness when incoherent data occur, whereas Coding Similarity almost gives constant reliability.

6.2 Iris

Table 3: Iris results

noise rate	Euclidean	Xing	ITML	CodeSim
0	0.9395	0.9646	**0.9837**	0.9652
5	0.9395	0.8233	**0.9825**	0.9013
10	0.9395	0.7173	**0.9672**	0.86
20	**0.9395**	0.6014	0.5214	0.7977
25	**0.9395**	0.5964	0.5964	0.7775
30	**0.9395**	0.5775	0.5514	0.7637

The results are written in table 3. The size of the small Iris dataset explains the abrupt decrease in similarity accuracy. Given the result of the Euclidean norm, using a learnt distance can be dangerous.

6.3 Wine

Table 4: Wine results

noise rate	Euclidean	Xing	ITML	CodeSim
0	0.7624	0.7694	0.8104	**0.9219**
5	0.7624	0.7697	0.6777	**0.8369**
10	0.7624	0.7682	0.7436	**0.7837**
20	0.7624	**0.775**	0.5753	0.711
25	0.7624	**0.779**	0.6698	0.6845
30	0.7624	**0.7789**	0.6075	0.6737

The results are written in table 4. Coding simlarity performs well with correct data. Surprisingly, Xing's algorithm gives the best results with really bad data. However, the accuracy is too close to the one from the Euclidean distance.

6.4 WDBC

Table 5: WDBC results

noise rate	Euclidean	Xing	ITML	CodeSim
0	0.8204	0.8358	**0.8567**	0.7231
5	0.8204	**0.8291**	0.7381	0.7012
10	**0.8204**	0.8199	0.634	0.6864
20	**0.8204**	0.795	0.7425	0.6659
25	**0.8204**	0.7852	0.6408	0.6594
30	**0.8204**	0.7766	0.6764	0.6527

The results are written in table 5. This dataset seems diffucult because neither the euclidean or the learnt distances give good results. In this case, using the Euclidean distance is the safest way to evaluate similarity.

6.5 Soybean-small

Table 6: Soybean-small results

noise rate	Euclidean	Xing	ITML	CodeSim
0	0.9417	0.9999	**1**	**1**
5	0.9417	0.9187	**1**	0.9539
10	0.9417	0.9146	**0.999**	0.8388
20	0.9417	0.8008	**0.9836**	0.7527
25	**0.9417**	0.7271	0.9274	0.7468
30	**0.9417**	0.6487	0.7849	0.7183

The results are written in table 6. The Euclidean distance gives good results enough to consider using another norm, which can behave pretty badly.

6.6 Balance-scale

Table 7: Balance-scale results

noise rate	Euclidean	Xing	ITML	CodeSim
0	0.6583	0.4146	**0.7771**	0.7476
5	0.6583	0.3946	0. 5135	**0.7356**
10	0.6583	-	0.5126	**0.7225**
20	0.6583	-	0.5269	**0.7012**
25	0.6583	-	-	**0.6914**
30	0.6583	-	-	**0.6848**

The results are written in table 7. With many wrong data, Xing's algorithm is unable to converge (at least in a decent time or number of iterations), such as ITML. This can be a result of the size of the dataset.

7. ANALYSIS OF RESULTS

First of all, with good data, the distance from the chosen algorithms almost always gives better results than the Euclidean distance. This confirms (if needed) the results published by their authors.

On small datasets (Iris, Soybean small), ITML performs well. However, the gain compared to the Euclidean distance is low. The tradeoff between a good distance and a safe one matters. In these case, perhaps the safest distance is the Euclidean one.

If we don't know anything about the set, the Euclidean norm can give random results. The Coding Similarity algorithm performs well in most cases and shows reasonable reliability. It has also the advantage to compute the distance without iterations. The process is really fast and cannot be caught in an infinite optimization loop.

8. CONCLUSIONS

The main result is this study that data drive the accuracy of the algorithms. No algorithm tends to dominate the other ones. Futhermore, results on well-defined sets may not represent the behavior on human-built ones. Who should define the similarity other than users when no class exists?

Because of possible errors in real datasets, one would choose an algorithm which shows a good robustness to the data's inconsistencies. However and once again, the data seem to decide what is the best algorithm.

The method to inject incoherent data can be discussed. We thought it was a fast and good way to simulate partially bad datasets. How bad can be the result on a similarity set which was created from by human being? This is really difficult to evaluate. The similarity may not be understand as a binary evaluation (similar / not similar). It can also not be seen as a quadratic function. The similarity evaluation errors can follow a pattern and not be totally distributed at random. Also these "choices" may be personal. If the training set was created by a unique user, the distance matrix reflects his definition of the similarity. Data can be close or far in a continuous way. Futhermore, this study is limited to Mahanalobis distances. Maybe a good "distance" is one which is not quadratic.

The result of some algorithms should not discourage to use them. If coding similarity performs well, it can be used to learn a good distance function and one can adjust A with new data captured on-the-fly with for example the online version of ITML.

So the difficulty to learn a good distance should not prevent from trying to use it. There is also a room left for further experiments with human-built datasets and non-quadratic distances. It may have many application in future recommendation processes, especially on-the-fly ones.

9. ACKNOWLEDGMENTS

We thank Sony Japan for this research carried out during an internship.
Financially supported by ANR-07-BLAN-0328-01 GAIA (Computational Information Geometry and Applications) and DIGITEO GAS 2008-16D (Geometric Algorithms & Statistics)

10. REFERENCES

[1] A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a mahalanobis metric from equivalence constraints. *J. Mach. Learn. Res.*, 6:937–965, 2005.

[2] L. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. In *USSR Computational Math. and Math. Physics*, 1967.

[3] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 209–216, New York, NY, USA, 2007. ACM.

[4] A. Globerson and S. Roweis. Metric learning by collapsing classes, 2005.

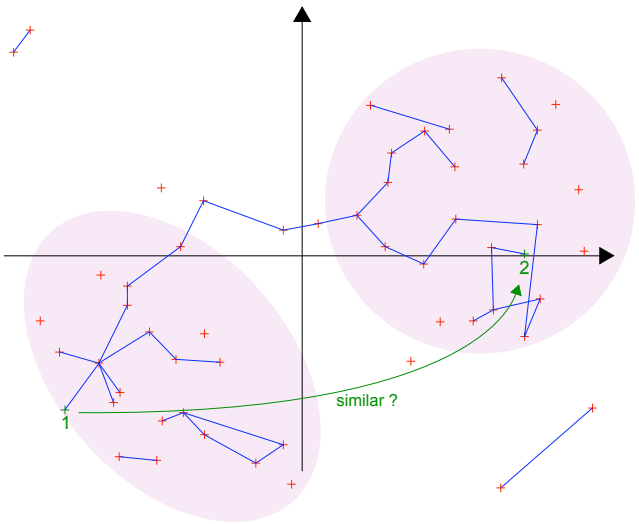

Figure 5: similarity closure

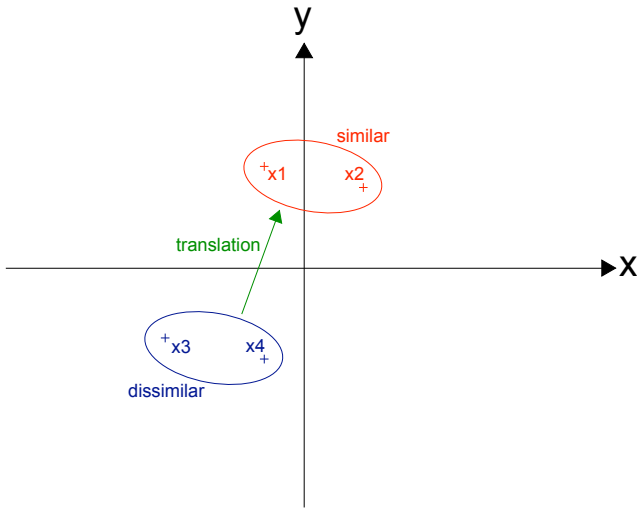

Figure 6: The distance between the points of each pair are the same, even if their similarity is not

[5] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2004.

[6] A. B. Hillel and D. Weinshall. Learning distance function by coding similarity. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 65–72, New York, NY, USA, 2007. ACM.

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

[8] L. Yang and R. Jin. An efficient algorithm for local distance metric learning. In *Proceedings of AAAI*, 2006.

APPENDIX

A. OTHER ALGORITHMS STUDIED

Many algorithms focus on looking for a distance. However, many of them can't be applied in our scenario. Here are some of the interesting ones we studied but at the end didn't use.

A.1 Learning a Mahalanobis Metric from Equivalence Constraints

This simple algorithm[1] does not require a large amount of equivalence constraints since it creates them. However, they are created from initial small groups of points (called *chunklets*) and these groups are extended thanks to the transitive closure of each. However, if there is no label on the points, the groups are not well defined. The transitive closure can reach the entire set(Fig. 5)... In our datasets, these labels exists but in music space, they are too fuzzy.

In fact, this algorithm is strongly related to *Coding Similarity* algorithm which is only an extension.

A.2 Neighbourhood Component Analysis

The problem formulation is very interesting here[5]. The goal is not to move closer similar points and separate dissimilar ones, but the goal is directly one application of learning a distance: maximize the k-Nearest Neighbour cross-validation accuracy. This is close to the traditional formulation of distance learning but this is not identical. However, labeled data are required, which does not fit our model.

A.3 Metric Learning by Collapsing Classes

This algorithm's goal[4] is to find the closest distance matrix to an ideal distance d_0 which perfectly separates the points. With a chosen Mahanalobis matrix A, we can define the distance $d^A_{i,j} = d_A(x_i, x_j) = \|x_i - x_j\|^2_A$ and the distribution $p^A(j|i) = \frac{e^{d^A_{i,j}}}{\sum_{k \neq i} e^{d^A_{i,k}}}$.

$$\min_{A \succeq 0} \sum_i \mathrm{KL}\left[p_0(j|i) \| p^A(j|i)\right]$$

$$\text{where } p_0(j|i) = \begin{cases} 0 & \text{if } (x_i, x_j) \in \mathcal{S} \Leftrightarrow d_0(x_i, x_j) = 0 \\ 1 & \text{if } (x_i, x_j) \in \mathcal{D} \Leftrightarrow d_0(x_i, x_j) = \infty \end{cases}$$

This algorithm is supposed to used labeled data, however simple similarity constraints are enough. However, several precision errors[10] makes it difficult to use.

A.4 An Efficient Algorithm for Local Distance Metric Learning

This paper[8] is perhaps one of the most interesting we came across, but it is at the same time one of the most difficult. The purpose is to locally learn a distance to prevent unsolvable cases such as translated points (Fig. 6). In this last figure, each pair has the same distance between its points. If one is similar, the other is dissimilar, it becomes impossible to solve the learning problem. This algorithm wasn't used because of lack of time but could give interesting results.

[10] $\sum_{k \neq i} e^{d^A_{i,k}}$ often exceeds `double` precision